UDC 004.056.5:004.415.5 **doi: 10.32620/reks.2026.2.06**

Artem ANTONENKO[1], Oleh BONDARENKO[2], Oleksandr GOLUBENKO[2], Natalia LASHCHEVSKA[1], Olga MOROZOVA[3]

[1]*National University of Life and Environmental Sciences of Ukraine, Kyiv, Ukraine*
[2]*Higher Education Institution "Academician Yuriy Bugay International Scientific and Technical University", Kyiv, Ukraine*
[3]*National Aerospace University "Kharkiv Aviation Institute", Kharkiv, Ukraine*

DEVSECOPS PIPELINE FOR PROVIDING SECURITY OF IOT CLIENT INTERFACES

This article examines *the processes for ensuring the information security of client interfaces (Front-end) in Internet of Things (IoT) systems.* ***This study aims to develop and implement*** *an automated DevSecOps pipeline integrated into the Gulp build environment to detect and block code vulnerabilities at the earliest stages of development (the Shift Left concept).* ***The tasks*** *are as follows: 1) analyze technological opportunities and challenges of implementing DevSecOps for IoT web interfaces; 2) analyze possible threats and vulnerabilities inherent in client-side code; 3) analyze existing approaches to integrating SAST tools into automated workflows; 4) analyze options for using standard automation tools (such as Gulp) to solve security tasks; 5) propose a DevSecOps pipeline architecture for the security of client interfaces; 6) propose a sequence of critical components and experimentally verify the effectiveness of the proposed solution. Based on the set tasks, the following results were obtained. An analysis of security problems in modern JavaScript-based IoT interfaces was conducted, and the need for automated code control was substantiated. An architecture for a DevSecOps pipeline is proposed using Gulp.js as an automation tool and a configured ESLint as a SAST scanner. A Gulp plugin ("wrapper") has been developed and implemented, providing continuous code monitoring during development. The system successfully identifies dangerous patterns (e.g., the use of eval functions or vulnerable setTimeout constructions) and blocks such code from entering the final release (Artifact). The proposed approach does not create significant overhead on the development process but guarantees compliance with security policies.* ***Conclusions.*** *The main contribution and scientific novelty of the results lie in creating an adaptive, easily integrated protection mechanism for the IoT ecosystem by integrating Static Application Security Testing (SAST) tools directly into the Gulp task runner architecture. Applying DevSecOps practices at the build level minimizes human-factor risks and increases overall trust in smart device management systems. The proposed solution is scalable and can serve as a foundational element of a cybersecurity strategy for Internet of Things projects.*

Keywords: *DevSecOps; Internet of Things (IoT); Gulp; Static Application Security Testing (SAST); pipeline; JavaScript; Cybersecurity Shift Left; CI/CD; Supply Chain Security*

Introduction

It is difficult to imagine the modern world without digital transformation, which is fundamentally changing approaches to work and interaction with technologies, requiring specialists to acquire new digital skills [1]. The number of connected devices generating and processing data in the Internet of Things (IoT) ecosystem is constantly growing, while issues of wireless sensor network coverage and security are becoming increasingly acute [2]. In this context, client web interfaces, especially those implemented as Single-Page Applications (SPAs), are becoming a critical entry point for managing complex systems, necessitating the integration of scientific approaches to their development and protection [3]. Ensuring cybersecurity is a crucial task for such solutions, and vulnerability research must be conducted throughout the lifecycle, including the code development stage [4].

The DevSecOps methodology is not merely another development model but acts as a "game changer" that integrates security practices directly into DevOps processes, although it is accompanied by several implementation challenges [5]. This innovative basis allows teams to leverage the power of cloud technologies (Cloud Native) and microservice architecture to create flexible and scalable solutions [6; 7]. This is achieved through the automation of development and deployment processes, where containerization and runtime environment security are key [8]. However, the cybersecurity of producing and using such systems throughout their lifecycle, especially for critical infrastructures, remains a subject of active research [9].

IoT services and Edge computing have opened new opportunities, transforming operations in many industries, from healthcare to smart cities, by moving computations closer to the data source [10; 11]. Companies can develop, test, and optimize microservices using cloud

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

platforms and edge computing, significantly reducing development time and costs, but this requires clear execution contracts and automation [12].

The adaptability of modern web interfaces, which can be updated in real-time, is particularly valuable for dynamic IoT tasks; however, this creates new threat vectors [13]. At the same time, cybersecurity issues must be carefully addressed, as code vulnerabilities such as XSS or injection attacks can have serious consequences; therefore, access control automation and script validation are necessary [14]. The "Shift Left" concept emphasizes the need for early threat detection because security begins with software development, and the cost of fixing an error grows exponentially as it progresses through later stages [15] (Fig. 1).

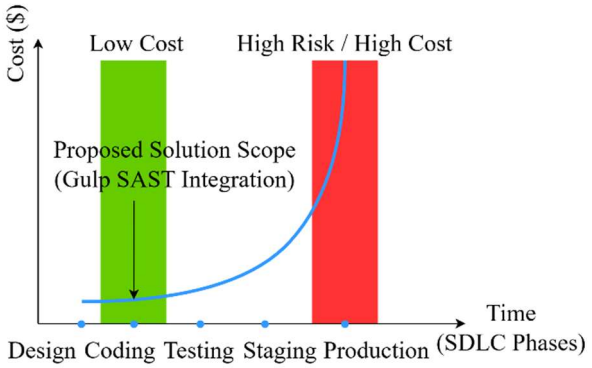

Fig. 1. Defect correction cost dependence on the SDLC stage

Like any resource on the World Wide Web, IoT management platforms are not only potential targets but also priority targets for attackers. The urgency of researching the problems of integrating automated security controls and identifying potential attacks cannot be overstated, as many practitioners still lack effective tools and an insufficient understanding of the DevOps landscape [16; 17]. Furthermore, the proactive identification of sophisticated attacks on IoT infrastructure requires advanced mechanisms for hypothesis generation and mathematical modeling. Adapting neural network mathematical models, which have proven to be highly effective for hypothesis generation in complex data arrays in other scientific domains [18], represents a critical vector for the future evolution of automated security controls.

This study aims to improve modern methods for ensuring the security of IoT system client interfaces by developing and integrating a DevSecOps pipeline for automated vulnerability detection and remediation, thereby increasing trust in the developed services.

The following tasks were considered and solved within the framework of this work to achieve this aim:

1) to analyze the technological opportunities and challenges of implementing DevSecOps for IoT web interfaces;

2) to analyze the possible threats and vulnerabilities inherent in client-side code in modern web applications;

3) to analyze existing approaches for integrating SAST tools into automated workflows [19];

4) to analyze options for using standard automation tools (such as Gulp) to solve security tasks, considering the problems of dependency "bloat" [20];

5) to propose a DevSecOps pipeline architecture for the security of client interfaces based on continuous security models [21];

6) to propose a sequence of critical components (rules and blocking mechanisms) to ensure the code build process's cybersecurity and experimentally verify the proposed solution's effectiveness.

This research is based on a combination of theoretical and empirical methods. Methods of system analysis and threat modeling were applied to analyze the specific nature of threats and existing approaches to securing IoT interfaces. The formalization method was used to transform abstract security standards (e.g., OWASP guidelines) into specific, automated blocking rules within the CI/CD environment. For the practical implementation of the proposed architecture, a prototyping method was utilized to develop a custom Gulp task runner integrated with a SAST scanner. The DevSecOps pipeline's effectiveness was validated through experimental testing on a dedicated testbed, employing quantitative measurement to assess the baseline architectural overhead and ensure the absence of significant delays in the continuous integration workflow.

1. Analysis of technological opportunities for DevSecOps implementation in IoT client interfaces

Considering the dynamics of modern web technology development, the implementation of client interfaces for Internet of Things (IoT) systems requires highly effective and secure solutions. Developers are increasingly using Cloud Native Engineering approaches [6] and microservice architecture [22; 7] to create scalable backends; however, the frontend part, implemented primarily as Single-Page Applications (SPAs), often remains a vulnerable point [3]. Ensuring cybersecurity at this level is a critical task because the interface is the entry point for both the user and a potential attacker.

The use of the DevSecOps (Development, Security, and Operations) methodology opens up opportunities for significantly increasing software security by integrating security checks directly into the Software Development Life Cycle (SDLC) [15]. This allows for solving security tasks effectively without slowing down code delivery processes (Continuous Delivery). However, unlike cloud solutions, where security tools are often provided by the platform, client-side development requires configuring local build pipelines.

Despite the availability of powerful CI/CD platforms, many teams face challenges when adapting them to the specific needs of IoT interfaces, particularly due to the lack of automated testing tools [23; 10]. The advantage of using lightweight task runners, such as Gulp, is the flexibility to configure data processing streams and parallel task execution. This allows Gulp to be viewed

not just as a minification tool but also as an effective environment for implementing SAST (Static Application Security Testing) [19].

Implementing a secure pipeline requires a clear sequence of actions, similar to the optimization of hardware solutions.

The first step in optimizing the development process is dependency management and combating their "bloat". Modern JavaScript packages (CommonJS/ESM) are prone to accumulating excess code, which not only affects performance but also increases the attack surface [20]. Using automated tools to analyze and clean package.json reduces the number of potentially vulnerable components even before the build begins. This is critically important, as supply chain and container vulnerabilities are among the main IoT threats [8].

The next step is to integrate the static analysis rules. The most important process is the transition from checking only code style (linting) to checking security patterns [13]. This involves configuring linters (e.g., ESLint) to detect JavaScript-specific vulnerabilities, such as code injections via eval() or dangerous DOM manipulations. Development tools automate this process, turning it into a routine operation that does not require deep cybersecurity knowledge from developers, thereby addressing the "skills gap" problem [1].

The next optimization stage is implementing "security gates" into the build process. This involves configuring the pipeline such that any violation of critical security rules leads to a forced build failure, which is a key requirement for security practitioners [16]. This allows the implementation of the Zero Trust concept [24; 7] regarding source code: no line of code should enter the final artifact without successful validation. This approach guarantees that vulnerable code cannot be physically deployed to user devices.

The last step is the formation of a secure artifact. At this stage, final code processing (minification and obfuscation) takes place, which complicates reverse engineering [4]. Optimizing the blocks and components from the previous steps yields a final application architecture that is resistant to known attack vectors and meets the performance requirements of Edge computing [10; 25].

To implement such an approach, standard languages and frameworks are used. The project can be implemented using the Node.js ecosystem, the standard for web development. Most libraries for solving IoT interface tasks are written in JavaScript. Therefore, the use of Gulp (based on Node.js) is a natural choice, as it allows the description of execution contracts and protection logic in the same language as the application itself [12].

If the project complies with the given requirements, the pipeline produces a verified, optimized package of files (HTML/CSS/JS). This is an intermediate layer that allows the interface to be securely integrated into more complex systems, such as Docker containers [8] or cloud environments [26].

Projects implemented with such a DevSecOps pipeline usually achieve a significantly higher level of compliance with security standards (e.g., the OWASP Top 10) without requiring additional security team resources, as confirmed by continuous security models [21]. This is a significant value, as it allows developers to focus on functionality while guaranteeing automated quality control.

The structure of all implementation levels of the proposed solution includes: the source code, static analysis (SAST), build automation (Gulp), and final artifact levels. The software part of the build service runs on the developer's machine or on CI server. It communicates with the file system to retrieve code, passes it through a set of rules (security "kernels"), and generates an error report or a finished build. All the software for accelerating security checks operates at this level.

2. Possible Threats and Vulnerabilities of IoT Client Interfaces

The architecture of modern Internet of Things (IoT) systems increasingly relies on complex web interfaces implemented as single-page applications (SPAs) to ensure effective and convenient user interaction with devices. However, every component of this ecosystem is subject to code-level security risks. The transition to distributed systems, including the possibility of multi-user management via the cloud or Edge gateways, requires an understanding of how individual client component vulnerabilities can affect the system as a whole [11].

The security of client-side code (JavaScript) executed in the user's browser is a critical issue, as any vulnerability can become an entry point for attacks on the physical IoT infrastructure. Since interface logic is often transmitted in clear text (or minified but reversible forms), it is critical to ensure the integrity of this code. Attackers may have many motives for manipulating client scripts: from intercepting access tokens to executing unauthorized commands on IoT devices [6; 7]. Considering that such interfaces are used in cyber-physical systems (e.g., in healthcare or smart cities), this can lead to negative physical consequences [24].

Software Supply Chain Attacks are one of the most serious threats to modern web applications. Modern Node.js projects use thousands of third-party packages (npm), which often contain redundant or "bloated" dependencies. Research shows that a significant portion of these dependencies is never used at runtime, yet they increase the attack surface and complicate auditing [20]. This is analogous to the introduction of hardware trojans, but at the software library level: malicious code can be hidden deep within the dependency tree and executed by the developer without notice.

The use of dangerous language constructs, such as dynamic code execution (eval), is another critical problem. This is a direct path to Cross-Site Scripting (XSS) – a vulnerability that allows attackers to inject malicious scripts into trusted web pages [13]. This is especially dangerous in the IoT context: via XSS, an attacker can hijack the session of a smart home administrator or an industrial controller. Despite the threat's notoriety (it consistently

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

ranks in the OWASP Top 10), developers continue to ignore secure coding practices and use dangerous functions to resolve tasks quickly [4].

Research into security configuration approaches is relevant when there is a threat of bypassing protection mechanisms due to human factors. Traditional control methods, such as manual Code Review or using IDE plugins, are ineffective because developers may ignore or disable them for the sake of development speed [23; 16]. Practitioner surveys show that only one-third of organizations use automated security testing tools, and a lack of knowledge remains a significant barrier [16].

In response to these challenges, the security paradigm is shifting towards the Zero Trust model ("trust no one") [24; 7]. This means that code cannot be considered safe by default because it was written by an internal developer. The continuous verification of every commit is necessary. Automated checks in CI/CD pipelines are becoming the standard for protection against insider threats and configuration errors, especially in cloud environments [26].

DevSecOps studies [21; 27] offer various models for integrating security, but they are often focused on the server-side or infrastructure, leaving client-side code insufficiently protected. Existing solutions for microservices and containers [22; 8] ensure isolation at the runtime level but do not prevent vulnerable code from being introduced at the build stage.

All the above confirm the relevance of researching the feasibility of applying established static analysis (SAST) approaches as automated barriers in the Gulp build process. The identified features will allow determining the most likely paths for violating the confidentiality, integrity, or availability of IoT interfaces. Information about known attack patterns and vulnerabilities must be systematized and transformed into automatic blocking rules [17]. Only a comprehensive approach that combines dependency optimization and strict syntax control can guarantee the service's necessary level of trust.

3. Analysis of the attack structure and features on IoT client interfaces

The architecture of IoT client interfaces, considered in the context of Cloud Native Engineering [6], comprises several interconnected layers. Based on this structure, possible threats for each system layer can be traced: – Web/Cloud Interface Layer – an online resource or PWA application through which the user interacts with the system; – CI/CD Environment Layer – a dedicated build server or cloud runner (e.g., the environment where Gulp is executed) that has access to the source code; – Application Artifact Layer – the compiled JavaScript bundle and HTML/CSS resources; – Dependency Layer – third-party libraries and modules (npm) integrated into the product.

Typically, the higher the abstraction level of the system, the easier it is to attack due to the presence of numerous logical vulnerabilities. In the system under consideration, the web service is at the top level, so the greatest attention is paid to attacks that can be implemented through client-side code.

Attacks on IoT interfaces can be diverse, considering the specifics of Edge computing and general web service vulnerabilities [10]. Several categories of attacks and methods of their implementation are worth highlighting:

1. Attacks on privacy. Since client interfaces process user data and access tokens to IoT devices, a high probability of sensitive information leakage exists. An attacker may attempt to intercept a session or API keys via Cross-Site Scripting (XSS) vulnerabilities if input data does not undergo proper sanitization during build or runtime [13]. This is especially critical for healthcare systems or smart homes, where privacy violations have direct consequences for user safety [11].

2. Attacks on integrity. Attackers may attempt to alter the interface logic by injecting malicious code into the software supply chain (Supply Chain Attack). This is achieved by compromising npm packages that the developer connects to the project. Without automated dependency control, the modified code enters the final bundle, allowing attackers to manipulate the physical device control functions [20].

3. Attacks on availability. Denial-of-service (DoS) attacks for IoT systems often operating under limited network resource conditions can be aimed at exhausting browser resources or the communication channel. The use of inefficient, "bloated" code or scripts that block the main execution thread can result in the inability to control the device at a critical moment [28; 2].

Like many other systems, the interface development environment may contain two types of vulnerabilities: software and infrastructure vulnerabilities. Software vulnerabilities are errors in proprietary code (e.g., using eval and innerHTML), outdated library versions, or improper input processing [4]. Infrastructure vulnerabilities may include security issues of the CI/CD pipeline itself, such as insufficient isolation of build containers or open access to configuration files [8; 19].

Vulnerability scanning is a mandatory stage of critical application testing. With the development of DevSecOps technologies, automation increases the quality of developed products [23]. In this regard, the possibility of detecting a critical vulnerability by automated means increases; however, in the "human-system" pair, the developer remains the weakest link.

If exploiting technical vulnerabilities is impossible due to robust Gulp pipeline filters, attackers can rely on the human factor. For example, developers may disable linter rules (suppress warnings) or ignore build errors to improve deployment speed [5]. Therefore, it is important that the security system is not only advisory but also blocking (fail-fast).

Note that the developer's machine or build server is part of the network infrastructure. Attacks on infrastructure can be carried out to redirect or substitute build artifacts,

thereby compromising the entire IoT service. This highlights the need to implement Zero Trust principles not only to the code but also to the environment in which the code is assembled [24; 7].

4. Analysis of approaches for integrating SAST into automated workflows

Integrating SAST into IoT interface development services enables detecting vulnerabilities at an early stage, even before the code is compiled into the final artifact. This includes analyzing syntactic constructions and project dependencies. The main stages of building such an automated protection process can be classified as follows [15]:

1. Pre-configuration. Defining security goals and testing boundaries. At this stage, the rules of the game are established: which code patterns are considered unacceptable. For IoT client interfaces, prohibiting the execution of arbitrary code (for example, via eval) and limiting the use of dangerous browser APIs are critical. The customer or security architect defines the configuration profile used as the standard. An example of such a configuration for the .eslintrc.json file, prohibiting dangerous eval constructions, is given in Code 1.

Possible performance issues are also considered at this stage: automated scanning should not significantly slow down the build time so as not to block the developers' work [23].

2. Context gathering. In the case of "White Box testing", which SAST is, the tool has full access to the source code. The project structure, directory tree, and manifest files (e.g., package.json) are analyzed at this stage. Understanding the configuration of the build system (in this case, Gulp) allows for the correct integration of the scanner into the data flow, as shown in the generalized scheme (Fig. 2). It is important to define root directories and exclusions (ignore files) so as not to waste resources on third-party libraries or test files that have already been verified by other means [19].

```
  "extends": [
    "eslint:recommended",
    "google"
  ],
  "rules": {
    // ... other style rules ...
    "no-implied-eval": "error", // Disallow implied eval
    "no-eval": "error"        // Disallow direct eval
  }
}
```

Code 1. Fragment of the .eslintrc.json file with added security rules

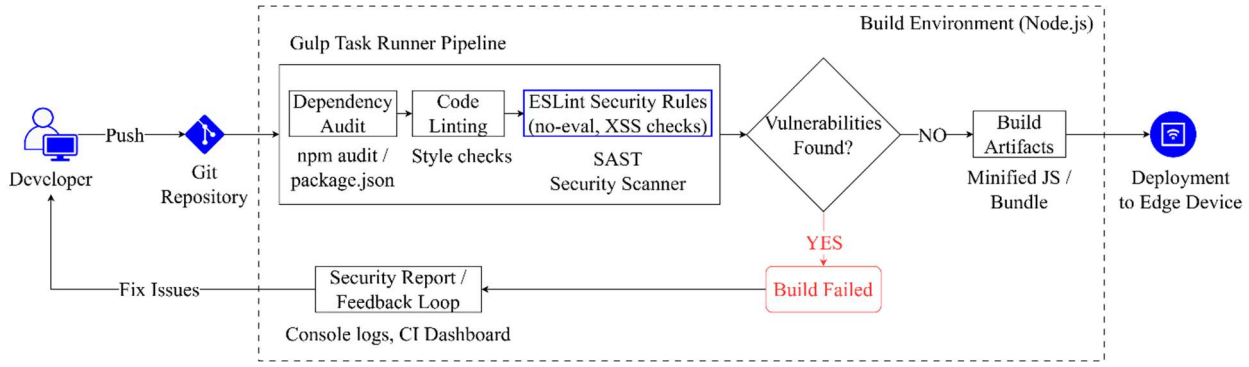

Fig. 2. Generalized architecture of the proposed DevSecOps pipeline with an automated feedback loop

3. Vulnerabilities identification. Automated tools (linters and scanners) are used to search for potential problems. Unlike manual testing, a pattern-matching mechanism is used here. Maximum attention is paid to the top level of the system's vulnerabilities, i.e., the web interface that interacts with the user. The tools are configured to search for matches with known vulnerability databases. To demonstrate the system's operation, a test file script.js was created containing intentionally introduced vulnerabilities, as shown in Code 2.

4. Blocking and Response. This is the key stage of DevSecOps, which distinguishes it from classical testing. Instead of "exploiting" the vulnerability, the system performs a preventive action, blocking the build process. If a critical error is found, the scanner generates a stop signal that interrupts the Gulp pipeline. Fig. 3 shows the implementation of this mechanism in the scriptLint Gulp task using the eslint.failAfterError() method. This ensures that compromised or low-quality code will not proceed to the next deployment stage [16; 29].

```
console.log("JS file is connected");
// For demonstrating XSS vulnerability
eval("console.log('This is a security risk!')"); // Error no-eval
// For demonstrating XSS vulnerability (implied eval)
setTimeout("console.log('Implied eval risk!')", 100); // Error no-implied-eval
// For demonstrating ESLint error (unused variable)
const unusedVariable = "This should cause an error"; // Error no-unused-vars
```

Code 2. Test file script.js with intentionally introduced errors

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

5. Reporting and providing feedback. The main goal of this stage is to provide the developer with instant information about the error. The report is directly output to the terminal console or the CI system log. It contains the following details: file name, line number, type of violated rule, and recommendations for correction. This allows the implementation of the fast feedback loop principle, improving developer qualifications without interrupting the workflow [5; 1].

```
const scriptLint = (env = "dev") => {
  let target;
  if (env === "src") {
    target = paths.src.js;
  } else if (env === "dev") {
    target = paths.copy.devJs;
  } else if (env === "dist") {
    target = paths.copy.distJs;
  }
  const startTime = Date.now();
  const processed = [];

  return (
    src(target)
      // .pipe(plumber())
      .pipe(eslint({ fix: true, cwd: root, useEslintrc: true }))
      .pipe(eslint.format())
      .pipe(eslint.failAfterError())
      .pipe(dest((file) => file.base))
      .on("data", (file) => processed.push(file))
      .on("end", () => {
        logTask({
          env,
          label: "Validation of JS files",
          files: processed,
          startTime,
          showSize: false,
        });
      })
  );
};
```

Fig. 3. Fragment of the Lint Gulp task script with integrated ESLint

The automated restart of verification after applying fixes (re-testing), which confirms the elimination of the vulnerability, is a possible next step. One option for using publicly available knowledge bases is to integrate rules corresponding to OWASP Top 10. The result of such adaptation is the development of an automated testing plan that covers specific threats for IoT interfaces [2].

While tool selection can be performed using recommendation systems, particularly those that use artificial intelligence [30], the basic level of protection must rely on deterministic, time-tested rules. A detailed integration plan for Gulp platforms should include testing aspects specific to the technology used (JavaScript/Node.js).

MITRE CWE knowledge base is an example of a source from which vulnerability descriptions are obtained. Hardware vulnerabilities are not as relevant as input validation and state management problems for JavaScript-oriented IoT interfaces. This knowledge must be used when creating the security rule configuration file.

Considering the specifics of Edge computing, where resources may be limited, the scanning process must be optimized [10]. The use of lightweight solutions integrated directly into the task runner avoids the need to deploy heavy scanners requiring separate infrastructure.

5. Proposed classification of DevSecOps pipeline components to ensure IoT security

The analysis of cybersecurity problems of IoT client interfaces shows that their protection can be viewed from different perspectives.

Computing nodes for code analysis. This is one form of service that can be used in cybersecurity analysis that requires significant computational power. In our case, the role of such a node is performed by the CI server or the local machine of the developer when running Gulp tasks. Such services can provide the ability to run checks specifically prepared for user tasks [23]. In this case, such a Gulp service is a tool for solving a specialized problem: finding vulnerabilities in the code even before its compilation. This allows the code to be processed much faster using resources that work better than traditional manual audit methods [15].

Implementation of cryptoprimitives and data protection. This is another example of using automated tools. The Gulp pipeline can be used to verify cryptographic correctness at the code-writing stage (for example, banning weak hashing algorithms via ESLint rules) [24]. Such verification can be performed on every commit (Continuous Integration), ensuring continuous verification. The ability to implement architecture for a specific task enables parallelizing linting and testing when data sets do not have dependencies. This also ensures the continuous operation of each security component, generating a verification result for every code change.

Physical implementation and device protection. This is an important and unique direction in which DevSecOps can be used as a service for IoT cybersecurity. Although we are talking about software code, it controls physical devices. A properly configured pipeline guarantees that the code reaching the device does not contain instructions that can damage the equipment or disrupt its operation (safety) [31]. The use of these implementation features prevents the possibility of emulating or spoofing control commands. This ensures cybersecurity components at the interface implementation level.

Vulnerability search in third-party components. The development of new IoT solutions often relies on ready-made libraries (e.g., npm). Such components contain third-party code that is inaccessible for a full manual audit. Only the vendor knows what is inside. Therefore, researching and analyzing the operation of such components (Supply Chain Security) is necessary for creating a secure service [20; 32]. A Gulp pipeline integrated with dependency audit tools (e.g., npm audit, Snyk) enables testing the expected behavior of these components for each new version.

Implementation of AI-based services This is another important and constantly developing area. Such services can be viewed as tools for performing code analysis tasks with reduced complexity and required effort [30]. However, they can also be viewed as research objects. Creating a service for a specific task and leveraging development environments enables prototyping specialized protection tools using languages and frameworks (e.g., Python and TensorFlow), thereby reducing labor costs during data analysis [33].

Comparing the proposed research steps with penetration tests helps us determine how to apply the proposed sequence to the study object. The comparison conducted from the perspective of the possibility of using the DevSecOps pipeline as a research object (see Table 1) allows concluding that the pipeline is only a tool for increasing code security in the first two use cases.

Table 1

Comparison of the proposed classification categories using the DevSecOps pipeline as a research object

DevSecOps pipeline purpose for solving cybersecurity tasks	**Pipeline as an object (target of attack/audit)**	**Pipeline as a tool (security control)**
Implementation of a computing node for accelerating code analysis (Linting/SAST)	No	Yes
Implementation of cryptographic primitive checks and data protection validation	No	Yes
Ensuring access control verification and identity management tasks	Yes	Yes
Search for vulnerabilities in third-party dependencies (supply chain)	Yes	Yes
Implementation of AI services for detecting anomalies in code	Yes	Yes

The difference is that in the first scenario, the end user (developer) prepares the entire project for Gulp-as-a-service for their build. However, the user can use a service with already prepared, well-tested rules (shared configs) to accelerate the implementation of cryptography checks.

The last three options consider the DevSecOps pipeline as both a research object and a tool for implementing important tasks. The option of ensuring access rights management uses the implementation specifics of a specific system instance. Here, the pipeline-as-object features, such as secrets management in CI/CD, enable identification of the instance and the exact service instance.

Researching third-party components (shell/dependencies) is an option where the project is loaded into the build system, which is a tool for researching the components themselves as an object. This analysis direction is critical for permitting the use of new library versions in services for critical systems.

Implementing AI using DevSecOps enables both the realization of a powerful service project with hardware-accelerated AI computations and the use of such a solution during the research of code vulnerabilities as an object. The third and last options are the most interesting directions for research and practical use, as they use implementation features unavailable to simple manual verification methods.

6. Proposed sequence of critical components for ensuring IoT client interface cybersecurity

As in many other areas of critical infrastructure, ensuring the cybersecurity of IoT platforms is a complex process, and ignoring any component can have serious consequences [9]. These critical security elements within the proposed DevSecOps pipeline are worth highlighting:

1. Regular dependency updates and remediation of supply chain vulnerability. The regular updating of npm packages and libraries is critical for maintaining web interface cybersecurity because it helps eliminate known vulnerabilities (CVE), improve functionality, and prevent supply chain attacks. In IoT projects, where client code often interacts with sensitive data, outdated libraries can become a weak link, exposing the system to unauthorized access [8]. Updates often include security patches discovered by the community after the release of previous versions. Furthermore, given the constant evolution of the cyber threat landscape, regular dependency auditing (e.g., via npm audit built into Gulp) helps adapt to new attack methods. Unupdated systems are not only prone to vulnerabilities but may also suffer from excess code ("bloat"), increasing the attack surface [20].

2. Continuous code change monitoring and analysis of build behavior. While hardware systems monitor traffic, monitoring every commit (Continuous Integration) is critical in software development. This allows for the rapid detection of anomalies, such as the sudden appearance of hid code or the disabling of linter rules, which may indicate an insider threat or a developer workstation compromise [21]. Effective monitoring involves using specialized tools (Gulp plugins) that analyze code in real time during the build process. Fig. 4 demonstrates

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

the result of the pipeline operation, which detected the code's vulnerabilities and blocked the build. This approach enables detection of attempts to inject malicious code before it reaches the repository or user devices [6]. The importance of this stage grows significantly in the IoT context, where attackers can leverage JavaScript's flexibility to rapidly propagate attacks.

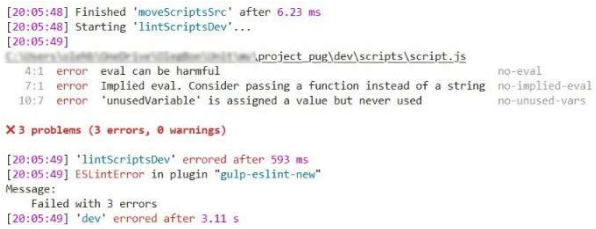

Fig. 4. Result of Gulp dev script execution with errors in the script.js file

3. Automated Security Gates instead of episodic audits. Regular security audits and penetration testing are important; however, in the modern pace of development, they must be supplemented by automated barriers. Integrating SAST into the Gulp pipeline allows identifying and fixing potential vulnerabilities before they are exploited [15]. These procedures include a comprehensive assessment of software aspects, including configuration analysis and code verification. Experienced specialists configure rules (e.g., banning eval), and automation ensures their strict execution. This helps protect against internal threats such as developer errors or insecure configurations [16]. This approach is a key element of the "Shift Left" strategy, which helps maintain customer trust in IoT services.

4. Compliance with security standards and best practices. When creating IoT services, compliance with standards is critical to ensuring their cybersecurity, especially in sensitive areas such as healthcare or smart cities [11]. Standards such as OWASP Top 10 and NIST recommendations provide frameworks for identifying and minimizing risks. The application of these standards through automatic ESLint rules helps ensure data confidentiality and integrity [13]. Moreover, compliance with international standards increases product competitiveness by demonstrating its high security and quality, a challenge for many organizations [23].

5. Increasing personnel awareness through tool feedback. Training personnel in protection methods plays a vital role, but in the DevSecOps context, tools become the "teachers". When the Gulp pipeline blocks a build and issues a clear error message (e.g., "Found potential XSS vector"), the developer learns instantly from their own experience [34]. When the vulnerabilities are eliminated, the pipeline successfully passes the code. Successful completion of the build after fixing errors is shown in Fig. 5.

Fig. 5. Result of Gulp dev script execution after fixing errors in the script.js file

Time metrics were collected during the build process of a basic IoT interface template to quantitatively evaluate the baseline performance overhead introduced by the proposed DevSecOps pipeline. As shown in the build statistics (Fig. 5), the total pipeline execution time for the test project (5 files, total size 8.8 Kb) was 2412 ms. The execution time for the lintScriptsDev task, which initializes the SAST scanner and acts as the security gate, was 457 ms. Since Gulp processes files in RAM using Node.js streams and the underlying abstract syntax tree (AST (Abstract Syntax Tree) traversal algorithms operate with linear O(N) time complexity, scaling to larger codebases (e.g., 10,000 lines of code) will only increase the execution time by fractions of a second. While the system's ability to automatically block vulnerable code patterns (as demonstrated in Fig. 4) proves the primary security effectiveness, these collected temporal metrics confirm that the proposed Shift-Left security layer introduces practically negligible architectural overhead, ensuring a fast feedback loop without slowing down the CI/CD process.

Developers who understand how attacks work and why certain code constructions are prohibited are an important element of the defense system. This helps minimize the risk of successful cyberattacks and strengthens the organization's overall cybersecurity strategy, helping bridge the digital skills gap [1]. Training contributes to the formation of a security culture, which is critical for maintaining a high level of protection against evolving threats.

7. Discussion

A study of the cybersecurity problems of IoT client interfaces was conducted, and a set of components for ensuring their security was proposed. Additionally, an

analysis of current threats inherent to modern web applications used for managing smart devices was conducted. Applying secure development standards in the context of automated Gulp workflows was examined. Regular code auditing and automated testing were identified as critical elements of the cybersecurity strategy, helping to maintain user trust in IoT services [23].

A complex of components has been proposed to counter modern threats, including regular dependency updates, monitoring and analysis of build behavior, implementation of Security Gates, compliance with standards, and personnel training. The metrics proposed in the FOBICS framework can serve as quantitative indicators for evaluating the effectiveness of such measures, such as the time to detect vulnerabilities and the number of critical errors blocked prior to release [35].

Compared to standalone enterprise SAST solutions such as SonarQube or specialized engines such as Semgrep, the proposed Gulp-integrated approach offers distinct advantages for frontend-heavy IoT projects. While SonarQube provides comprehensive multi-language analysis and historical tracking, it requires the deployment and maintenance of a dedicated server infrastructure, which contradicts the lightweight and fast-paced nature of client-side development. Semgrep offers fast, rule-based scanning, but its integration often requires installing external binaries outside the Node.js ecosystem. By contrast, leveraging ESLint as a security scanner directly within the Gulp pipeline ensures zero infrastructure overhead. The analysis runs entirely in-memory during the build process using native JavaScript dependencies, providing developers with instant, local feedback without relying on external network requests or third-party computing resources.

Notably, the proposed approach has the potential to be applied not only in classic IoT systems but also in Digital Twin technologies, especially in the energy sector (Smart Grids). As researchers have noted, integrating AI and machine learning into big data management requires mechanisms for information collection [36]. Furthermore, the integrity of the collected data is a critical prerequisite for implementing advanced predictive models in cyber-physical systems, such as utilizing LSTM neural networks for dynamic environmental forecasting [37]. Since client interfaces often serve as tools for visualizing and managing digital twins, ensuring their integrity through the proposed DevSecOps pipeline is critical to preventing data manipulation at the "human-machine" level.

The scientific novelty of the results lies in investigating the feasibility of integrating static analysis (SAST) directly into lightweight automation tools (Gulp), with the client code of the IoT interface as the protected object. This concept builds upon recent successful practices of embedding SAST directly into Gulp pipelines to secure complex applied information systems without compromising build speed [38]. As in many other fields, ensuring IoT cybersecurity is a complex task, and ignoring any component (e.g., checking for "bloated" dependencies) can have serious consequences [20]. Reliance solely on manual testing is insufficient; therefore, a comprehensive list of measures for automating protection is provided, emphasizing the importance of transitioning from a reactive to a proactive approach in development.

Conclusions

The practical value of this study lies in offering a comprehensive cybersecurity strategy for developing IoT client interfaces. The proposed solution, which includes automated code auditing, blocking of dangerous constructions, and dependency management, enables the detection and elimination of vulnerabilities at the build stage, thereby increasing security and trust in the final product.

A study of cybersecurity problems in modern web interfaces for managing smart devices was conducted, and a set of components to address these problems was proposed. An analysis of current threats specific to the JavaScript environment was conducted. The application of the DevSecOps methodology to lightweight automation tools was examined. Regular static analysis and automated Security Gates were identified as key strategy elements that help maintain the integrity of the IoT ecosystem.

A set of critical components to address modern threats was proposed to ensure cybersecurity throughout the development process. The complex includes regular dependency updates, continuous code monitoring and analysis, automated auditing, compliance with security standards, and feedback mechanisms that increase personnel awareness.

The scientific novelty of the obtained results lies in the creation of an adaptive and easily integrated protection mechanism for the IoT ecosystem by integrating SAST tools directly into the Gulp task runner architecture as a protection object. This allows implementing the "Shift Left" concept for specific IoT tasks, ensuring code execution control on user devices.

Ensuring the cybersecurity of IoT client interfaces is a complex task, and ignoring any component can have serious consequences. Reliance solely on manual testing is insufficient; therefore, a comprehensive list of protection automation measures within the CI/CD pipeline is provided.

Future research will be dedicated to the following directions: (i) research and development of advanced static analysis techniques specifically adapted for IoT framework specifics, including automated tools for detecting complex logical vulnerabilities; (ii) study of the application of Artificial Intelligence (AI) and Machine

Learning (ML) for real-time code analysis, focusing on detecting anomalies in script behavior and predictive threat analytics; (iii) development of compatibility frameworks guaranteeing that DevSecOps pipelines can seamlessly integrate with existing Cloud and Fog Computing infrastructures, adhering to evolving regulatory requirements.

Contributions of authors: formulation of the aim and tasks, literature analysis, development of the Gulp pipeline architecture, conducting experiments, writing the draft of the article, formatting the list of references – **Oleh Bondarenko**; formulation of the concept, conducting experiments, validation of experiment results, translation – **Artem Antonenko**; methodology formulation, analysis of SAST integration approaches – **Oleksandr Golubenko**; general editing and reviewing – **Natalia Lashchevska**; literature search in databases, review and editing – **Olga Morozova**.

All authors have read and approved the final manuscript for publication.

Conflict of Interest

The authors declare that there is no conflict of interest regarding this research, whether financial, personal, authorship, or of any other nature, that could influence the research and its results presented in this article.

Data Availability

The data supporting the findings of this study are available within the article. The source code and configuration files used in the study are available from the corresponding author upon request.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence methods when creating the presented work.

References

1. Hoe, S. L. Digital transformation and the future of work: closing the digital skills gap. *Development and Learning in Organizations: An International Journal*, 2024, vol. 39, no. 3, pp. 14–17. DOI: 10.1108/DLO-06-2024-0167.

2. Moslehi, M. M. Exploring coverage and security challenges in wireless sensor networks: A survey. *Computer Networks*, 2025, vol. 260, article no. 111096. DOI: 10.1016/j.comnet.2025.111096.

3. Drane, L., McDonnell, M., Petras, R., Stiner, C., Ruckman, A. J., et al. Integrating scientific single-page applications with DevSecOps. *Future Generation Computer Systems*, 2025, vol. 166, article no. 107695. DOI: 10.1016/j.future.2024.107695.

4. Cope, R. Strong security starts with software development. *Network Security*, 2020, vol. 2020, no. 7, pp. 6–9. DOI: 10.1016/S1353-4858(20)30078-7.

5. Rajapakse, R. N., Zahedi, M., Babar, M. A., & Shen, H. Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology*, 2022, vol. 141, article no. 106700. DOI: 10.1016/j.infsof.2021.106700.

6. Brojabasi, S., Paul, S., & Mitra, A. Cloud native engineering: A comprehensive review of principles, practices, and challenges. *Advances in Computers*, 2026, vol. 141, pp. 331–353. DOI: 10.1016/bs.adcom.2025.06.013.

7. Shin, D., Kim, J., Pawana, I. W. A. J., & You, I. Enhancing cloud-native DevSecOps: A Zero Trust approach for the financial sector. *Computer Standards & Interfaces*, 2025, vol. 93, article no. 103975. DOI: 10.1016/j.csi.2025.103975.

8. Sroor, M., Mohanani, R., Colomo-Palacios, R., Dasanayake, S., & Mikkonen, T. Managing security issues in software containers: From practitioners' perspective. *Journal of Systems and Software*, 2026, vol. 231, article no. 112616. DOI: 10.1016/j.jss.2025.112616.

9. Tanque, M., & Foxwell, H. J. Cyber risks on IoT platforms and zero trust solutions. *Advances in Computers*, 2023, vol. 131, pp. 79–148. DOI: 10.1016/bs.adcom.2023.04.003.

10. Ranganath, S. Edge computing: Types and attributes. *Advances in Computers*, 2022, vol. 127, pp. 35–62. DOI: 10.1016/bs.adcom.2022.03.001.

11. Tawalbeh, L., Muheidat, F., Tawalbeh, M., Quwaider, M., & Abd El-Latif, A. A. Edge enabled IoT system model for secure healthcare. *Measurement*, 2022, vol. 191, article no. 110792. DOI: 10.1016/j.measurement.2022.110792.

12. Truong, H.-L., & Klein, P. DevOps Contract for Assuring Execution of IoT Microservices in the Edge. *Internet of Things*, 2020, vol. 9, article no. 100150. DOI: 10.1016/j.iot.2019.100150.

13. Rey Rodriguez, K. S., Avellaneda Galindo, J. D., Tárrega Juan, J., Bermejo Higuera, J. R., Bermejo Higuera, J., & Sicilia Montalvo, J. A. Secure Development Methodology for Full Stack Web Applications: Proof of the Methodology Applied to Vue.js, Spring Boot and MySQL. *Computers, Materials and Continua*, 2025, vol. 85, no. 1, pp. 1807–1858. DOI: 10.326204/cmc.2025.067127.

14. Nyarko-Boateng, O., Nti, I. K., Mensah, A. A., & Gyamfi, E. K. Controlling user access with scripting to mitigate cyber-attacks. *Scientific African*, 2024, vol. 26, article no. e02355. DOI: 10.1016/j.sciaf.2024.e02355.

15. Saeed, H., Shafi, I., Ahmad, J., Khan, A. A., Khurshaid, T., & Ashraf, I. Review of Techniques for Integrating Security in Software Development Lifecycle. *Computers, Materials and Continua*, 2025, vol. 82, no. 1, pp. 139–172. DOI: 10.326204/cmc.2024.057587.

16. Sinan, M., Shahin, M., & Gondal, I. Implementing and integrating security controls: A practitioners' perspective. *Computers & Security*, 2025, vol. 156, article no. 104516. DOI: 10.1016/j.cose.2025.104516.

17. Zhang, X., Zhao, P., & Jaskolka, J. Navigating the DevOps landscape. *Journal of Systems and Software*, 2025, vol. 223, article no. 112331. DOI: 10.1016/j.jss.2024.112331.

18. Zaitsev, I., Golubenko, O., Tkachenko, O., Pidmohylnyi, O., & Antonenko, A. Exploring advanced hypothesis generation in astronomy through the implementation of a mathematical model of linguistic neural networks. *CEUR Workshop Proceedings*, 2023, vol. 3687, pp. 121–128.

19. Casola, V., De Benedictis, A., Mazzocca, C., & Orbinato, V. Secure software development and testing: A model-based methodology. *Computers & Security*, 2024, vol. 137, article no. 103639. DOI: 10.1016/j.cose.2023.103639.

20. Liu, Y., Tiwari, D., Bogdan, C., & Baudry, B. Detecting and removing bloated dependencies in CommonJS packages. *Journal of Systems and Software*, 2025, vol. 230, article no. 112509. DOI: 10.1016/j.jss.2025.112509.

21. Kumar, R., & Goyal, R. Modeling continuous security: A conceptual model for automated DevSecOps using open-source software over cloud (ADOC). *Computers & Security*, 2020, vol. 97, article no. 101967. DOI: 10.1016/j.cose.2020.101967.

22. Rezaei Nasab, A., Shahin, M., Hoseyni Raviz, S. A., Liang, P., Mashmool, A., & Lenarduzzi, V. An empirical study of security practices for microservices systems. *Journal of Systems and Software*, 2023, vol. 198, article no. 111563. DOI: 10.1016/j.jss.2022.111563.

23. Akbar, M. A., Smolander, K., Mahmood, S., & Alsanad, A. Toward successful DevSecOps in software development organizations: A decision-making framework. *Information and Software Technology*, 2022, vol. 147, article no. 106894. DOI: 10.1016/j.infsof.2022.106894.

24. Ren, Y., Wang, Z., Sharma, P. K., Alqahtani, F., Tolba, A., & Wang, J. Zero Trust Networks: Evolution and Application from Concept to Practice. *Computers, Materials and Continua*, 2025, vol. 82, no. 2, pp. 1593–1613. DOI: 10.326204/cmc.2025.059170.

25. Robitzsch, S., Centenaro, M., di Pietro, N., Cordeiro, L., Gomes, A. S., Sanders, P., & Ishaq, A. Prospects on the adoption of a microservice-based architecture in 5G systems and beyond. *Computer Networks*, 2023, vol. 237, article no. 110058. DOI: 10.1016/j.comnet.2023.110058.

26. Casino, F., Lopez-Iturri, P., & Patsakis, C. Cloud continuum testbeds and next-generation ICTs: Trends, challenges, and perspectives. *Computer Science Review*, 2025, vol. 56, article no. 100696. DOI: 10.1016/j.cosrev.2024.100696.

27. Olotu, S. I., Oronti, A. O., & Alese, B. K. Microsegmentation for containerized micro-services in edge computing. *Intelligent Data-Centric Systems: Cybersecurity Defensive Walls in Edge Computing*, Academic Press, 2026, pp. 85–104. DOI: 10.1016/B978-0-443-34109-0.00011-5.

28. Kadri, M. R., Abdelli, A., Ben Othman, J., & Mokdad, L. Survey and classification of DoS and DDoS attack detection and validation approaches for IoT environments. *Internet of Things*, 2024, vol. 25, article no. 101021. DOI: 10.1016/j.iot.2023.101021.

29. Khandebharad, A. Migration From DevOps to DevSecOps. *International Journal of Cloud Applications and Computing*, 2022, vol. 12, no. 1, pp. 1–12. DOI: 10.4018/IJCAC.2022010102.

30. Khadem, E. A., & Movaghar, A. From challenges to metrics: An LLM-driven DevOps recommendation system grounded in evidence-based mappings. *Array*, 2025, vol. 28, article no. 100547. DOI: 10.1016/j.array.2025.100547.

31. Oruma, S. O., Sánchez-Gordón, M., & Gkioulos, V. Enhancing security, privacy, and usability in social robots: A software development framework. *Computer Standards & Interfaces*, 2026, vol. 96, article no. 104052. DOI: 10.1016/j.csi.2025.104052.

32. Schuh, G., Jarke, M., Gützlaff, A., Koren, I., Janke, T., & Neumann, H. Review of commercial and open technologies available for Industrial Internet of Things. *Design and Operation of Production Networks for Mass Personalization in the Era of Cloud Technology*, 2022, pp. 209–241. DOI: 10.1016/B978-0-12-823657-4.00005-1.

33. Silva, C., Felisberto, J., Barraca, J. P., & Salvador, P. ASAP 2.0: Autonomous & proactive detection of malicious applications for privacy quantification in 6G network services. *Computer Communications*, 2025, vol. 237, article no. 108145. DOI: 10.1016/j.comcom.2025.108145.

34. Ascenção, C., Teixeira, H., Gonçalves, J., & Almeida, F. Large-scale agile security practices in software engineering. *Information and Computer Security*, 2024, vol. 33, no. 3, pp. 344–361. DOI: 10.1108/ICS-07-2023-0136.

35. Caniglia, A., Dentamaro, V., Galantucci, S., & Impedovo, D. FOBICS: Assessing project security level through a metrics framework that evaluates DevSecOps performance. *Information and Software Technology*, 2025, vol. 178, article no. 107605. DOI: 10.1016/j.infsof.2024.107605.

36. Djebali, S., Guerard, G., & Taleb, I. Survey and insights on digital twins design and smart grid's applications. *Future Generation Computer Systems*, 2024, vol. 153, pp. 234–248. DOI: 10.1016/j.future.2023.11.033.

37. Makoveichuk, O., Golubenko, O., Kukhtyk, S., Antonenko, A., Bereznychenko, V., & Iatsyshyn, A. Temperature Forecasting with LSTM: A Case Study on Kyiv Weather Data. *CEUR Workshop Proceedings*, 2025, vol. 4133, pp. 201–211.

38. Zaitsev, I., Bondarenko, O., Golubenko, O., Antonenko, A., & Savchenko, A. Securing applied information systems with SAST integration into the Gulp pipeline. *CEUR Workshop Proceedings*, 2025, vol. 4133, pp. 181–189.

Received 27.01.2026, Received in revised form 25.03.2026
Accepted date 15.07.2026, Published date 31.07.2026

DEVSECOPS-ПАЙПЛАЙН ДЛЯ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ КЛІЄНТСЬКИХ ІНТЕРФЕЙСІВ СИСТЕМ IOT

Антоненко А. В., Бондаренко О.В., Голубенко О. І., Лащевська Н. О., Морозова О. І.

Предметом *дослідження статті є процеси забезпечення інформаційної безпеки клієнтських інтерфейсів (Front-end) у системах Інтернету речей (IoT).* ***Метою*** *даної роботи є розробка та практична реалізація автоматизованого DevSecOps-пайплайну, інтегрованого в середовище збірки Gulp, для виявлення та блокування вразливостей коду на ранніх етапах розробки (концепція Shift Left).* ***Завдання****: 1) проаналізувати технологічні можливості та виклики впровадження DevSecOps для веб-інтерфейсів IoT; 2) проаналізувати можливі загрози та вразливості, притаманні клієнтському коду; 3) проаналізувати існуючі підходи до інтеграції інструментів статичного аналізу (SAST) в автоматизовані потоки робіт; 4) проаналізувати варіанти використання стандартних інструментів автоматизації (таких як Gulp) для вирішення завдань безпеки; 5) запропонувати архітектуру DevSecOps-пайплайну для безпеки клієнтських інтерфейсів; 6) запропонувати послідовність критичних компонентів та експериментально перевірити ефективність запропонованого рішення. За результатами виконаних завдань отримано такі результати. Проведено аналіз проблем безпеки в сучасних JavaScript-орієнтованих інтерфейсах IoT та обґрунтовано необхідність автоматизованого контролю коду. Запропоновано архітектуру DevSecOps-пайплайну, що використовує Gulp.js як інструмент автоматизації та налаштований ESLint як SAST-сканер. Розроблено та*

програмно реалізовано Gulp-плагін («обгортку»), який забезпечує безперервний моніторинг коду під час розробки. Експериментально доведено, що система успішно ідентифікує небезпечні патерни (наприклад, використання функцій eval або вразливих конструкцій setTimeout) та блокує потрапляння такого коду у фінальний реліз (Artifact). Продемонстровано, що запропонований підхід не створює значного навантаження на процес розробки, але гарантує дотримання політик безпеки. ***Висновки.*** *Основний внесок і наукова новизна отриманих результатів полягає у створенні адаптивного та легко інтегрованого механізму захисту для екосистеми IoT шляхом інтеграції інструментів статичного аналізу (SAST) безпосередньо в архітектуру таск-ранера Gulp. Показано, що застосування підходів DevSecOps на рівні збірки дозволяє мінімізувати ризики людського фактору та підвищити загальну довіру до систем управління розумними пристроями. Запропоноване рішення є масштабованим і може бути використане як базовий елемент стратегії кібербезпеки в проєктах Інтернету речей.*

Ключові слова: *DevSecOps; Інтернет речей (IoT); Gulp; Static Application Security Testing (SAST); pipeline; JavaScript; кібербезпека;-Shift Left; CI/CD; безпека ланцюжка постачання*.

Антоненко Артем Васильович – кандидат технічних наук, доцент, доцент кафедри стандартизації і сертифікації с.г. продукції, Національний університет біоресурсів і природокористування України, м. Київ, Україна.

Бондаренко Олег Валерійович – здобувач вищої освіти кафедри інформаційних та комунікаційних технологій, ЗВО «Міжнародний науково-технічний університет імені академіка Юрія Бугая», м. Київ, Україна.

Голубенко Олександр Іванович – кандидат технічних наук, доцент, завідувач кафедри інформаційних та комунікаційних технологій, ЗВО «Міжнародний науково-технічний університет імені академіка Юрія Бугая», м. Київ, Україна.

Лащевська Наталія Олександрівна – к.т.н., доцент, доцент кафедри комп'ютерних наук, Національний університет біоресурсів і природокористування України, Київ,Україна

Морозова Ольга Ігорівна – доктор технічних наук, проф., проф. каф. кібербезпеки та інтелектуальних інформаційних технологій, Національний аерокосмічний університет «Харківський авіаційний інститут», м. Харків, Україна.

Artem Antonenko – Ph.D. in Technical Sciences, Associate Professor, Associate Professor at the Department of Standardization and Certification of Agricultural Products, National University of Life and Environmental Sciences of Ukraine, Kyiv, Ukraine.
e-mail: artem.v.antonenko@gmail.com, ORCID: 0000-0001-9397-1209, Scopus Author ID: 57207861964

Oleh Bondarenko – Bachelor's Student, Department of Information and Communication Technologies, Higher Education Institution "Academician Yuriy Bugay International Scientific and Technical University", Kyiv, Ukraine.
e-mail: olegbon.ua@gmail.com, ORCID: 0009-0008-2516-143X

Oleksandr Golubenko – PhD in Technical Sciences, Associate Professor, Head of the Department of Information and Communication Technologies, Higher Education Institution "Academician Yuriy Bugay International Scientific and Technical University", Kyiv, Ukraine.
e-mail: o.golubenko@istu.edu.ua, ORCID: 0000-0002-1776-5160, Scopus Author ID: 57552544800

Nataliia Lashchevska – PhD, Associate Professor, Associate Professor at the Department of Computer Science, National University of Life and Environmental Sciences of Ukraine, Kyiv, Ukraine.
e-mail: n.lashchevska@duikt.edu.ua, ORCID: 0000-0003-2148-115X, Scopus Author ID: 56125845600

Olga Morozova – Doctor of Technical Science, Professor, Professor at the Department of Cybersecurity and Intelligent Information Technologies, National Aerospace University «Kharkiv Aviation Institute», Kharkiv, Ukraine,
e-mail: o.morozova@csn.khai.edu, ORCID: 0000-0001-7706-3155, Scopus Author ID: 57194517520.