Static analysis of Elixir web application with RefactorErl

Smiljana Knežev[1,*], István Bozó[1] and Melinda Tóth[1]

[1]ELTE, Eötvös Lóránd University, Faculty of Informatics, Budapest, Hungary

Abstract

The quality of a software product's source code can be evaluated from various perspectives. Today, one of the most crucial quality measures is security. Ideally, the number of security vulnerabilities should be zero. Detecting security vulnerabilities in software can be done either dynamically or statically. The latter approach helps uncover issues during development before the product is released. Static analysers become particularly crucial when creating secure and reliable web applications. Therefore, our research focuses on the static analysis of Elixir web applications. The biggest challenge when using static analysis tools is delivering precise and useful results without many false-positive hits. This paper builds on our previous research and examines the application of RefactorErl in accurately identifying vulnerabilities in Elixir web applications. The previous research focused on developing algorithms for checking vulnerabilities for the common web application vulnerabilities according to the Erlang Ecosystem Foundations. It showed that semantic filtering and data-flow analysis reduce the number of false positive results. This work continues to check the feasibility of static analysis and develop analysis for the rest of the security vulnerabilities according to the Erlang Ecosystem Foundation's recommendations. Compared with previous research, we note that the most vulnerabilities present in a project belong to the common web application vulnerabilities. Nonetheless, several other vulnerabilities, including session vulnerabilities, TLS vulnerabilities, and information leakage, were found. There was no overlap in the vulnerabilities we observed in this paper with the tools we compared them to. The analysis was done on five open-source projects.

Keywords

secure coding, Elixir, vulnerability identification, static analysis

1. Introduction

Static analysis plays an essential role in ensuring the safety, reliability, and correctness of software. Tools that use static analysis are widely adopted and used in the industry.

Erlang and Elixir programming languages run on the BEAM virtual machine and are used for building scalable, reliable, and fault-tolerant systems. They provide the underlying technology that ensures essential services run smoothly. Major platforms, including WhatsApp, Discord, RabbitMQ, and banking systems for instant payments, all utilise BEAM to manage millions of requests and a vast number of transactions daily. These requests can pose serious threats, putting customers at risk of data theft or loss and potentially causing significant damage to the system. When such systems fail, it impacts entire sectors. Secure coding helps mitigate the harm caused by such attacks. By using code analysis, developers can enhance the security of their applications and mitigate potential risks.

One of the biggest challenges in providing secure code analysis is providing precise and useful results. Some vulnerabilities can be quickly identified with a simple text-based analysis, while a syntactic tree analysis approach can provide more accurate results. However, both methods can still produce many false positives. Too many results mean developers have to spend a significant amount of time manually reviewing them, which can lead to mistakes. The semantic analysis approach can help lower the number of false positives.

Erlang Ecosystem Foundation (EEF) [1] offers guidelines for developing secure BEAM applications. Inside those guidelines, recommendations for developing web applications in BEAM languages can be

SQAMIA 2025: Workshop on Software Quality, Analysis, Monitoring, Improvement, and Applications, September 20–12, 2025, Maribor, Slovenia

*Corresponding author.

t16wzn@inf.elte.hu (S. Knežev); bozoistvan@elte.hu (I. Bozó); tothmelinda@elte.hu (M. Tóth)

https://www.inf.elte.hu (S. Knežev); https://www.inf.elte.hu (I. Bozó); https://www.inf.elte.hu (M. Tóth)

0009-0006-2947-0390 (S. Knežev); 0000-0001-5145-9688 (I. Bozó); 0000-0001-6300-7945 (M. Tóth)

found. These recommendations primarily pertain to web applications built using the Phoenix framework [2] or simple Plug applications [3].

RefactorErl [4] is a static code analysis and transformation tool for Erlang. A security analyser framework for Erlang is built on top of RefactorErl. This framework was extended with Elixir analysis. In our previous work [5], we supported the analysis of the most common web application vulnerabilities based on the EEF guidelines. Furthermore, we compared our analysis of open-source projects with Sobelow, showing a reduction in false-positive results. In this paper, we focus on the precise implementation and evaluation of the remaining EEF guidelines.

The rest of the paper is structured as follows. Section 2 gives a short outline of static analysis tools used in secure coding and mentions previous research on the subject. Section 3 gives an overview of all security guidelines for web applications by EEF and provides more detail on the vulnerabilities to be analysed. In section 4, we describe how Elixir code is evaluated in RefactorErl. Finally, sections 5 and 6 present results of security analysis on open-source projects and conclude the paper.

2. Related work

Service Organization Control 2 (SOC 2) [6], Open Worldwide Application Security Project (OWASP) [7], and National Institute of Standards and Technology (NIST) [8] are well-known but distinct cybersecurity standards and frameworks that serve complementary purposes. SOC 2, developed by the American Institute of Certified Public Accountants (AICPA), is an audit standard that evaluates how customer data is protected based on the Trust Services Criteria: Security, Availability, Processing Integrity, Confidentiality, and Privacy. It does not focus on specific technologies but on whether security and operational controls are well-designed (Type I) and consistently effective over time (Type II). SOC 2 is relevant for SaaS companies, organizations handling sensitive data, and cloud providers.

OWASP is an open, community-driven resource for improving software security. It provides practical tools, such as the OWASP Top 10, the Application Security Verification Standard (ASVS), and the Software Assurance Maturity Model (SAMM), which help organizations assess and improve security practices across the software development lifecycle. OWASP is implementation-focused, providing actionable methods to write and maintain secure code.

NIST publishes a broad set of cybersecurity frameworks and technical standards. The NIST Cybersecurity Framework (CSF) is a widely adopted, risk-based model suitable for organizations of all sizes. NIST's SP 800 series provides detailed, structured guidance for building and managing security programs. In practice, these three fit together. NIST sets the strategic risk management and governance structure, OWASP ensures application-level security in development, and SOC 2 provides independent confirmation that security controls are in place and working. As the need for security analysis grew, tools for different programming languages were developed. Some of the most popular are SonarQube [9], SpotBugs [10], and CodeChecker [11], which offer analysis for several programming languages. Some tools specialise in one language or a family of languages, such as ESLint [12] for JavaScript, Bandit [13] for Python, and Brakeman [14] for Ruby on Rails.

Sobelow [15] is a security static analysis tool for Elixir and Phoenix. Sobelow analyses Elixir's abstract syntax trees. Currently, the tool supports analysis for Insecure configuration, Known-vulnerable Dependencies, Cross-Site Scripting, SQL injection, Command injection, Code execution, Denial of Service, Directory traversal, and Unsafe serialisation. The tool favours over-reporting rather than under-reporting, resulting in false-positive hits.

Semgrep [16] is a static analysis security tool that supports multiple languages, including Elixir. It utilises pattern matching and data-flow analysis encapsulated in rules, which can be either predefined or custom-made.

These tools are either standalone analysis tools or support integration with the most commonly used IDEs. The goal of these tools is to identify security issues early in the software development phase, thereby saving time and money in project development. They are often used as a part of the CI/CD pipelines, making vulnerability detection available with every build.

If we examine research on Elixir's vulnerability, there is an empirical study covered by D. B. Bose et al. [17] that analyses vulnerability-related commits in Elixir projects. In the study, 4,446 commits from 25 open-source projects were analysed, and 2% were vulnerability-related. In 9 out of 25 projects, at least one security vulnerability was found. The study concluded that further research is necessary and that the use of static code analysis tools could yield more detailed results.

In our previous work [5], we analysed common web application vulnerabilities from the Erlang Ecosystem Foundation's best practices for Secure web applications running on BEAM recommendations. We found that applying semantic filtering and data-flow analysis reduces false positive results in the security analysis of Elixir projects.

3. Secure Elixir web applications

Recommendations in the EEF security guidelines for web applications running on BEAM [1] are divided into five categories: "Common Web Application Vulnerabilities", "Session Management Vulnerabilities", "TLS Vulnerabilities", "Information Leakage", and "Supply Chain Vulnerabilities" as shown in Table 1. Each category contains descriptions of several vulnerabilities listed in the second column of the Table 1. The last column of the table shows the status of the implementation within RefactorErl's static analysis environment.

In our previous work [5], we developed algorithms for the security analysis of the "Common Application Vulnerabilities". More precisely, for "Cross-Site Scripting (XSS)", "Cross-Site Request Forgery (CSRF)", "Cross-Site WebSocket Hijacking (CSWSH)", "SQL Injection", and "Denial of Service (DoS)". "Client-side enforcement of server-side security" was not implemented because it contained general recommendations and was not found appropriate for the static code analysis.

"Supply Chain Vulnerabilities" are not suitable for static analysis in RefactorErl because their detection already relies on the use of community-maintained tools or tools that use Software Bill-of-Materials (SBoM). Hex [18], a package manager for the Elixir/Erlang community, helps detect outdated, retired, and changed packages. Detecting changed packages through manual review helps prevent "Insufficient due diligence with 3rd-party components". For "Dependencies with known vulnerabilities," either the `mix deps.audit`, an audit tool that checks known vulnerabilities by comparing them to the known vulnerabilities in the Elixir Security Advisories repository [19], or with some other tools that use SBoM are used.

In this paper, we focus on analysing "Session Management Vulnerabilities", "TLS Vulnerabilities", and "Information Leakage" [7, 8, 20, 21].

3.1. Session Management Vulnerabilities

Session Management Vulnerabilities [7, 8, 20, 21] allow an attacker to gain unauthorised access to user sessions, impersonate users, and misuse their privileges. This can lead to data theft, account takeover, fraudulent transactions, or other unauthorised actions.

EEF categorises these vulnerabilities in the following groups: "No server-side session revocation", "No server-side session timeout", "Session leakage (session hijacking)", "Session fixation", "Session information leakage", and "Session lifecycle and WebSocket connections".

The `mix phx.gen.auth` tool is an official generator included with the Phoenix framework for adding user authentication to web applications. It provides secure defaults for password hashing, secure cookies, and other related features. It also avoids authentication mistakes, such as storing plain-text passwords, and saves development time. This tool is recommended in several sections of the "Session Management Vulnerabilities" category of the EEF guidelines.

No server-side session revocation: Plug/Phoenix cookie-based session store data on the client side. This doesn't allow session revocation on the server side. If the attacker obtains the session cookie, they can reuse it even after the user logs out. Secure session cookie management implies both the use of client-side cookies and server-side cookies tracking. The auth tool `mix phx.gen.auth` is

Table 1
List of the EEF vulnerabilities and the status of the checkers

Category	Vulnerability	Status
Common Web Application Vulnerabilities		
	Cross-Site Scripting (XSS)	[5]
	Cross-Site Request Forgery (CSRF)	[5]
	Cross-Site WebSocket Hijacking (CSWSH)	[5]
	SQL Injection	[5]
	Denial of Service (DoS)	[5]
	Client-side enforcement of server-side security	–
Session Management Vulnerabilities		
	No server-side session revocation	–
	No server-side session timeout	–
	Session leakage (session hijacking)	✓
	Session fixation	✓
	Session information leakage	✓
	Session lifecycle and WebSocket connections	–
TLS Vulnerabilities		
	Insufficient peer certificate verification	✓
	Weak TLS versions, ciphers, and other options	✓
	Misconfigured TLS offload	✓
	Downgrade attacks	✓
Information Leakage		
	Elixir and Erlang standard library	–
	Parameter filter in Phoenix logger	✓
	Phoenix socket messages	–
	Redacting fields in Ecto schemas	–
Supply Chain Vulnerabilities		
	Outdated dependencies	–
	Retired dependencies	–
	Dependencies with known vulnerabilities	–
	Insufficient due diligence with 3rd-party components	–
	Insufficient visibility into 3rd-party components	–

recommended to generate appropriate session lifecycle handling. Since this is a tool recommendation for usage during project development, it is not feasible for static analysis in RefactorErl.

No server-side session timeout: Server-side session timeouts are essential because we cannot always rely on cookie expiration to close the session. Users can forget to log out, or their browsers may crash during a session, so server-side session tracking is crucial to prevent security risks. Decreasing the default time that `mix phx.gen.auth` sets and adding mechanisms for sooner session invalidation are recommended. As in the previous case, this is not feasible for static analysis in RefactorErl.

Session leakage (session hijacking): To prevent cookie leakage, which results in session leakage (session hijacking) [20, 21], several server-side configurations are recommended. `HttpOnly` and `Secure` attributes should be set when setting a cookie.

Session fixation: A type of attack where the attacker plants a session ID for the user to use, allowing the attacker to hijack the user's account after login [21]. The attack can be prevented by rotating the session ID after authentication. The `Plug.Session` API provides `configure_session/2` function for renewing the session ID. Projects that use `mix phx.gen.auth` have a `renew_session/1` function for rotating session IDs.

Session information leakage: Session information leakage could be prevented by encrypting the contents of a cookie [20, 21]. Encryption can be enabled by setting the `encryption_salt`

in `Plug.Session`. For non-session cookies, the `encrypt: true` option should be passed to `Plug.Conn.put_resp_cookie/4`.

Session lifecycle and WebSocket connections: WebSocket connections established during a session should be disconnected when the session is revoked. The `mix phx.gen.auth` has mechanisms that employ this, and the custom solutions should also have this in mind. Since this recommendation involves tool usage or building a custom solution, it is not feasible for static analysis in RefactorErl.

3.2. TLS Vulnerabilities

TLS vulnerabilities are associated with the misconfiguration of the Transport Layer Security (TLS) protocol [20, 21]. This can lead to compromised confidentiality and integrity of the data transferred over HTTPS or other TLS channels.

Insufficient peer certificate verification: A vulnerability that occurs when server certificates are not properly verified when making HTTPS requests to external APIs [21].

In these scenarios, attackers can intercept, alter, or eavesdrop on sensitive data exchanged between the application and its intended peer, compromising both data integrity and confidentiality. Therefore, robust peer certificate verification is essential to ensure secure, authenticated communication channels and prevent these classes of attacks.

Weak TLS versions, cyphers and other options: Outdated and weak TLS versions, cyphers, and options make servers vulnerable [20, 21]. Following up-to-date recommendations [7, 8] when configuring TLS is recommended. Also using `Plug.SSL.configure/1` function or `cipher_suite` option in the endpoint's https configuration is advised.

Misconfigured TLS offload: When the TLS is not terminated by the Phoenix or Plug app, the client request arrives at the app as plain HTTP. The `Plug.Conn` structure used in the Phoenix app contains a field that shows if the request arrived over HTTP or HTTPS. Setting this field properly is important in preventing security vulnerabilities. Some of the recommended options include using `force_ssl` in the Endpoint's configuration or `Plug.SSL` with `rewrite_on`.

Downgrade attacks: In a downgrade attack, an attacker attempts to force a user to access a site over the HTTP protocol instead of the more secure HTTPS protocol [20, 21]. By doing this, an attacker can intercept the request and potentially expose users' data and credentials. To protect against downgrade attacks, the server must send the HTTP Strict-Transport-Security (HSTS) header to the browser. In Phoenix, this is done by setting the `hsts` option to `true` in `Plug.SSL` or setting the `force_ssl` in the endpoint's configuration to true.

3.3. Information Leakage

Information leakage [20, 21] typically occurs through events such as application logs, debugging output, verbose error messages, or the use of verbose inspection tools. Sensitive information, including passwords, authentication tokens, secret keys, or personally identifiable user data, may be unintentionally recorded or displayed during routine system processes. Such disclosures can be exploited by attackers or unauthorised users, resulting in unauthorised access, privilege escalation, or other system compromise. The mechanisms recommended by the Erlang Ecosystem Foundation aim to systematically reduce the risk of information leakage by advocating for strategies, such as masking or redacting sensitive data prior to logging. Adoption of these techniques enables developers to minimise the risk of confidential data exposure and supports compliance with security and data requirements.

Elixir and Erlang standard library: The Elixir standard library and security guidelines outline tools and methods for preventing the leakage of secrets. One method is to implement the Inspect protocol that redacts all sensitive information. Other methods recommend using the `redact: true` option on Ecto schema fields or wrapping sensitive information in closures. These recommendations

rely on the developer knowing which information is sensitive and protecting it with the mentioned methods. Therefore, this is not feasible for static analysis with RefactorErl.

Parameter filter in Phoenix logger: `Phoenix.Logger` provides configurable mechanisms for filtering sensitive parameters from request logs in Phoenix applications. Developers can specify, within the application's configuration files, a list of parameter keys whose values should be redacted before logging, thereby preventing the accidental exposure of sensitive data, such as passwords or tokens, in logs.

Alternatively, Phoenix supports an allow-list approach, where only explicitly permitted parameters are included in the logs and all others are automatically filtered. This flexible filtering capability enables applications to better protect user privacy and comply with security best practices.

Phoenix socket messages: In Phoenix LiveView apps, if the socket process dies unexpectedly while data is being sent over it, the message triggering error is logged in the crash report, and parameter filtering doesn't help. The recommendation to prevent this is to implement the `Inspect` protocol to redact data before logging it. This recommendation is not feasible for static analysis in RefactorErl, for there is no one way to implement the protocol.

Redacting fields in Ecto schemas: To protect sensitive data being shown in database models, sensitive fields in an Ecto schema can be defined with the `redact: true` option. In this way, the fields will be replaced with redacted data, preventing accidental leaks.

4. Methods

We are using static analysis to identify vulnerable source code fragments. We build an intermediate representation from the Elixir source code at first and run our analysis on that. We defined and implemented several algorithms to identify the security issues.

RefactorErl [4] is a static analysis and transformation tool for Erlang. It utilises the Semantic Programming Graph (SPG) [22] as an intermediate representation of source code. SPG is built from Erlang's Abstract Syntax Tree (AST) and extended with a set of static analysers to add semantic information. It is possible to build various static analysis and refactoring functionalities on top of the SPG [23]. RefactorErl also provides data– and control–flow analysis [4]. RefactorErl supports building SPG from both Erlang source files and compiled beam files. Since Elixir is compiled to the same Abstract Format as Erlang, it is possible to build SPG from it. Therefore, existing analysers can be used on Elixir programs, and designing new, custom-tailored ones for Elixir's specific needs can be done. A framework for security analysis was built on top of SPG [24, 25]. In our previous work [5], we developed several security analysers for Elixir's web applications. In the rest of this section, we describe some of the algorithms that extend the framework with missing Elixir security analysers for web applications.

The pseudo-code for detecting a session-hijacking vulnerability is given in Algorithm 1. The helper functions used in this algorithm are provided in the Algorithm 2 and in the Algorithm 3.

In the Algorithm 1, we look for `put_resp_cookie` function calls. When we find them, we should examine their fourth argument, which is a list of options that contains tuples. That list of tuples should contain `{http_only, true}`, `{secure, true}` values.

Function ANALYZEFUNS will run data-flow analysis on the provided arguments and return them in the form of `{Origin, {Parameter, DataFlow}}`. When we receive those results, we filter out only the option tuples and search for `{http_only, true}`, `{secure, true}` values within them. For those filtered two-element option tuples, we run data-flow analysis first, comparing it to the desired value, and after that, we repeat the same process for the second element of the option tuple, as shown in Algorithms 3 and 2.

Other session management and information leakage-related security checkers follow a similar detection pattern. Typically, we search for various initial functions and filter them based on different options.

Algorithm 1 Detect calls vulnerable to session–hijacking

```
function GetCallsForSessionHijackingVulnerability(Fun)
    SessionFunctions ← {(Elixir.Plug.Conn, [put_resp_cookie])}
    DataFlowTuples ← AnalyzeFuns(Fun, SessionFunctions, GetChildrenForFourth)
    OptionTuples ← []
    for all (O, (_, D)) ∈ DataFlowTuples do
        for all E ∈ D do
            OptionTuples ← OptionTuples ∪ {(O, GetOptionTuples(E))}
        end for
    end for
    OptionTuplesFiltered ← {(O, E) | (O, E) ∈ OptionTuples ∧ E ≠ ∅}
    SafeOptions ← {(http_only, true), (secure, true)}
    VulnerableEntries ← []
    for all (O, E) ∈ OptionTuplesFiltered do
        for all SO ∈ SafeOptions do
            if IsOptionUnsafe(E, SO) then
                VulnerableEntries ← VulnerableEntries ∪ {O}
                break
            end if
        end for
    end for
    return VulnerableEntries
end function
```

Algorithm 2 Checks if provided option (tuple) is safe

```
function IsOptionUnsafe((Ch1, Ch2), (Option, Value))
    return HasValue(Ch1, Option) and HasValue(Ch2, Value)
end function
```

Algorithm 3 Runs data-flow analysis on argument and compares its value

```
function HasValue(Expr, Value)
    Df ← GetOrigins({(Expr, [Expr])}, {(back, true)})
    ExprValues ← []
    for all (_, (_, D)) ∈ Df do
        for all E ∈ D do
            ExprValues ← ExprValues ∪ {Expr:Value(E)}
        end for
    end for
    return Member(Value, ExprValues)
end function
```

The TLS-related vulnerabilities are primarily the result of incorrect or incomplete configurations. The pseudocode of a representative algorithm (Misconfigured TLS offload) for analysing configuration files is shown in Algorithm 4. Here, no data-flow analysis is used, but a simple check (`CheckSslForce`) is made to see if the recommended arguments from the EEF guidelines are set.

Algorithm 4 Checks configuration files for force_ssl

1: **function** GetMissconfiguredOfloadExs(Fun)
2: $Config \leftarrow \{$(Elixir.Config, [config])$\}$
3: $FunExpressions \leftarrow$ GetCallsFor(Fun, $Config$)
4: $TupleExpression \leftarrow$ GetTupleExpressions($FunExpressions$)
5: **return** $\{E \mid E \in TupleExpression,$ CheckSslForce$(E)\}$
6: **end function**

5. Results

Refactorerl tool [26] can be run as a standalone tool or can be integrated into a CI/CD pipeline. The tool currently runs on Windows, Linux, and macOS. Erlang/OTP 25 or newer is required for security analysis [**?**].

To evaluate the methodology described in Section 4, we analysed some frequently used open-source projects. The analysis is built on algorithms described in [5]. We concluded that there was no need to apply data-flow analysis to configuration files, as they primarily contain specifications and configuration details. In these cases, simply checking if an option is provided was sufficient.

We analysed the following open-source projects:

- Hexpm [27] is a repository website that hosts packages used by Hex, Elixir's package manager.
- Changelog [28] is an open-source podcast software that provides content for developers.
- Plausible Analytics [29] is an open-source alternative to Google Analytics that prioritises privacy.
- Authex [30] is an example of an open-source plug application. It is a simple API for an OAuth2/OpenID server.

Hexpm, Changelog, and Analytics are Phoenix applications, which are actively used and maintained. Authex was chosen as an example of a Plug web application. Table 2 shows the size in terms of lines of code of each of the analysed projects.

Time required for a less demanding vulnerability-detecting algorithm, like insufficient peer certification verification on a mid-size project, e.g., Hexpm, takes about 8 seconds.

Results of the analysis were compared to Sobelow's results.

Table 2
Projects overview

Project	Loc
Hexpm	14159
Changelog	11422
Analytics	76094
Authex	1942

5.1. Session Management Vulnerability analysis

From the session management vulnerabilities, we identified session hijacking, session fixation, and session information leakage, all of which are suitable for static analysis. We evaluated these vulnerabilities in open-source projects using RefactorErl. Table 3 presents the evaluation results. Sobelow does not offer a session management vulnerability analysis.

5.2. TLS Vulnerabilities

In RefactorErl analysis of TLS vulnerabilities, we note that every project has misconfigured TLS offload, which means that `force_ssl` is not enabled anywhere in the configuration files. Additionally, we

Table 3
Analytics session vulnerabilities

Vulnerability	Num
Session hijacking	3
Session fixation	0
Session information leakage	3

frequently find downgrade attack vulnerabilities in every project, indicating that `hsts: true` is not enabled in the endpoint configuration. We found no instances of insufficient peer certificate verification or weak TLS versions, cyphers, and other options in any of the projects, as shown in Table 4.

Table 4
TLS vulnerabilities results

Vulnearability	Hexpm	Analytics	Changelog	Authex
Insufficient peer certificate verification	0	0	0	0
Weak TLS versions, ciphers, and other options	0	0	0	0
Misconfigured TLS offload	1	1	1	1
Downgrade attacks	2	2	1	1

Regarding TLS vulnerabilities, Sobelow marks HTTPS Not Enabled, which is equivalent to misconfigured TLS offload with high confidence. It also marks HSTS Not Enabled with medium confidence. HSTS Not Enabled checks if inside of `https` configurations `force_ssl` is enabled. RefactorErl checks for all occurrences of `force_ssl` in all configuration files for downgrade attacks.

In Table 5, we can see that the results match those of misconfigured TLS offload in RefactorErl's analysis, except for the Authex project. This is because Sobelow checks this vulnerability only in `prod.exs`, but in the Authex `prod.exs` file, it does not exist. We also note that the HSTS Not Enabled vulnerability is present only in the Analytics project.

Table 5
Sobelow TLS vulnerabilities results

Vulnearability	Hexpm	Analytics	Changelog	Authex
Config.HTTPS: HTTPS Not Enabled	1	1	1	0
Config.HSTS: HSTS Not Enabled	0	1	0	0

5.3. Information Leakage

Only the Parameter filter in the Phoenix logger is feasible for reliable static code analysis (Table 6). Other vulnerabilities depend on which fields users find vulnerable.

Table 6
Information leakage results

Vulnerability	Hexpm	Analytics	Changelog	Authex
Parameter filter in Phoenix logger	1	2	1	1

Sobelow reports hard-coded secrets. This is a recommendation from OWASP [7] that outlines the general EEF guidelines and should be incorporated into the RefactorErl analysis in future work. Hardcoded secrets were only found in the Hexpm project as shown in Table 7. Sobelow marks this vulnerability as high confidence.

Table 7
Hard-coded secrets in Sobelow results

Vulnearability	Hexpm	Analytics	Changelog	Authex
Config.Secrets: Hardcoded Secret	3	0	0	0

5.4. Discussion

In our previous work [5], we demonstrated that a semantic-based approach to security analysis and data-flow analysis can reduce the number of false-positive results in common web application vulnerabilities compared to an AST-based analysis, such as in Sobelow.

This paper aimed to investigate the feasibility of implementing an analysis of the rest of the EEF security guidelines for web applications and to compare it to an AST-based approach again.

For the evaluated vulnerabilities, we found that session management vulnerabilities, session hijacking, and session information leakage were only present in one of the four analysed projects. Sobelow lacks corresponding vulnerability detection, preventing us from comparing our results.

When it comes to TLS vulnerabilities, Sobelow supports HTTPS Not Enabled and HSTS Not Enabled checks, which correspond to misconfigured TLS offload and downgrade attacks in the EEF guidelines, respectively. We observe that the results partially match. This is because Refactorerl considers all configuration files, whereas Sobelow restricts its analysis to a subset.

Also, in case of session fixation, Insufficient peer certificate verification, and Weak TLS versions, ciphers, and other options, there were no vulnerabilities found.This suggests further analysis on a larger number of projects is needed.

The parameter filter in the Phoenix logger vulnerability was found at least once in every project. This analysis is not present in Sobelow. However, Sobelow checks for hardcoded secrets, found in the Hexpm project, where Refactorerl is missing an implementation.

Manual check did not find any false positive results on the analysed projects. Therefore, in analysing these projects, we can only compare the analysis coverage. Further analysis is needed to investigate the influence of these approaches on false-positive reduction.

Sobelow analysis is notably faster, but lacks the precision and detail of the semantic-oriented approach we use in Refactorerl. In our future work, we aim to optimize the execution time of the vulnerability analysis.

6. Conclusions

Static analysis of web applications plays a key role in developing a safe application. It facilitates the early detection of security vulnerabilities while the software is still in the development phase. As a result, potential losses in time and resources are drastically reduced. For even more efficient use of these tools, precise and comprehensive analysis that yields few false-positive results is needed. To achieve this, RefactorErl uses semantic filtering and data-flow analysis. Since Elixir and Erlang run on the same virtual machine, BEAM, we analyse the compiled Elixir Web applications by loading beam files into RefactorErl.

In our previous work, we initiated a secure analysis of Elixir web applications in accordance with the EEF guidelines for the most common web vulnerabilities. We showed that the use of data-flow analysis reduces false-positive results. This was confirmed through an analysis of open-source projects and a comparison with Sobelow.

This work aims to provide a comprehensive analysis of the EEF guidelines, with a focus on implementing the missing vulnerability analysis. Implemented and evaluated vulnerability categories, including session management vulnerabilities, TLS vulnerabilities, and information leakage. Missing vulnerabilities from the EEF guidelines were categorised as not feasible for static code analysis because they were general and non-specific recommendations that could not be uniquely identified in the source

code.

These analysers were evaluated on the four open-source projects and compared with the results of the SAST tool. It was found that Sobelow does not support session management vulnerability analysis, and the only results of these vulnerabilities using RefactorErl were found in the Analytics project. In project analysis, three instances of session hijacking and three instances of session information leakage were found for the Analytics project.

Regarding TSL vulnerabilities, Sobelow supports two analysers while RefactorErl supports four. In the Sobelow analysis, the HTTPS Not Enabled vulnerability, which is equivalent of a misconfigured TLS offload in Refactorerl, matches for all projects except Authex. Sobelow's HSTS Not Enabled is reported only once, just for the Analytics project. Refactorerl reports this vulnerability as misconfigured TLS offload for all projects once. Further, Refactorerl analysis showed that Hexpm and Analytics have two Downgrade attack vulnerabilities, and Changeog and Authex have one.

When it comes to information leakage, these two tools have disjoint analyzers. Sobelow follows OWASP recommendations, which will be part of future work in RefactorErl analysis.

Future work will include an analysis of additional projects and the refinement of security analysis for web applications in Elixir. We will cover other guidelines, such as OWASP and NISR, and add developer usability studies for a more comprehensive analysis. In our previous work, we noted that our method marks unknown functions as vulnerable, even if they are part of a standard Elixir library. To prevent this, we need to develop an understanding of frequently used library functions in Elixir. Furthermore, as the analysis improves, we plan to develop a transformation tool for web application vulnerabilities within the RefactorErl framework.

Acknowledgments

Project no. TKP2021-NVA-29 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] Erlang Ecosystem Foundation (EEF), Security Working Group, Web Application Security Best Practices for BEAM languages, 2025. URL: https://erlef.github.io/security-wg/web_app_security_best_practices_beam/index, accessed: 29-04-2025.

[2] G. Lessel, Pheonix in Action, Manning, ISBN 978-1617295041, 2019. URL: https://www.manning.com/books/elixir-in-action-third-edition.

[3] Phoenix Community, Plug documentation, 2025. URL: https://hexdocs.pm/phoenix/plug.html, accessed: 29-04-2025.

[4] M. Tóth, I. Bozó, Static analysis of complex software systems implemented in Erlang, Central European Functional Programming Summer School – Fourth Summer School, CEFP 2011, Revisited Selected Lectures, Lecture Notes in Computer Science (LNCS), Vol. 7241, pp. 451-514, Springer-Verlag, ISSN: 0302-9743, 2012.

[5] S. Knežev, I. Bozó, M. Tóth, Identifying security issues in Elixir web applications, in: Programming 2025, OpenAccess Series in Informatics (to appear in), 2025.

[6] American Institute of Certified Public Accountants, SOC 2, 2025. URL: https://www.aicpa-cima.com/topic/audit-assurance/audit-and-assurance-greater-than-soc-2, accessed: 11-08-2025.

[7] OWASP Foundation, Application Security Verification Standard, 2025. URL: https://owasp.org/www-pdf-archive/OWASP_Application_Security_Verification_Standard_4.0-en.pdf, accessed: 29-04-2025.
[8] National Institute of Standards and Technology, Cypersecurity Framework, 2025. URL: https://www.nist.gov/cyberframework, accessed: 11-08-2025.
[9] G. A. Campbell, P. P. Papapetrou., SonarQube in Action, Manning Publications, 2013.
[10] SpotBugs, Documentation, 2025. URL: https://spotbugs.readthedocs.io/en/latest/introduction.html, accessed: 29-04-2025.
[11] CodeChecker, Documentation, 2025. URL: https://codechecker.readthedocs.io/en/latest/, accessed: 29-04-2025.
[12] OpenJS Foundation and ESLint contributors, ESLint, n.d. URL: https://eslint.org/, accessed: 29-04-2025.
[13] PyCQA (Python Code Quality Authority), Bandit, 2025. URL: https://bandit.readthedocs.io/en/latest/, accessed: 29-04-2025.
[14] Brakeman, Ruby on Rails Static Analysis Security Tool, 2025. URL: https://brakemanscanner.org/, accessed: 29-04-2025.
[15] NCC Group, Sobelow, 2025. URL: https://github.com/nccgroup/sobelow, accessed: 29-04-2025.
[16] Semgrep, Inc., Semgrep, 2025. URL: https://github.com/semgrep/semgrep, accessed: 29-04-2025.
[17] D. B. Bose, K. Cottrell, A. Rahman, Vision for a secure Elixir Ecosystem: An empirical study of vulnerabilities in Elixir programs, in: Proceedings of the 2022 ACM Southeast Conference, ACM SE '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 215–218. URL: https://doi.org/10.1145/3476883.3520204. doi:10.1145/3476883.3520204.
[18] Hex Team, Hex, 2025. URL: https://hex.pm/, accessed: 29-04-2025.
[19] Mirego, Elixir security advisories, 2025. URL: https://github.com/mirego/elixir-security-advisories, accessed: 29-04-2025.
[20] R. Anderson, Security Engineering: A Guide to Building Dependable Distributed Systems, 3rd ed., Wiley, 2020.
[21] I. Ristić, Bulletproof TLS and PKI, 2nd ed., Feisty Duck, 2022.
[22] Z. Horváth, L. Lövei, T. Kozsik, R. Kitlei, A. N. Víg, T. Nagy, M. Tóth, R. Király, Modeling semantic knowledge in Erlang for refactoring, in: Knowledge Engineering: Principles and Techniques, Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2009, volume 54(2009) Sp. Issue of *Studia Universitatis Babeş-Bolyai, Series Informatica*, Cluj-Napoca, Romania, 2009, pp. 7–16.
[23] I. Bozó, D. Horpácsi, Z. Horváth, R. Kitlei, J. Köszegi, T. M., M. Tóth, RefactorErl - source code analysis and refactoring in Erlang, in: Proceedings of the 12th Symposium on Programming Languages and Software Tools, ISBN 978-9949-23-178-2, Tallin, Estonia, 2011, pp. 138–148.
[24] M. Tóth, I. Bozó, Supporting secure coding for Erlang, in: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, SAC '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 1307–1311. URL: https://doi.org/10.1145/3605098.3636185. doi:10.1145/3605098.3636185.
[25] M. Tóth, Don't let it crash - How we Applied our Security Checks on Elixir Code, Talk at ELixirConf, Lisbon, 2024, 2024. URL: https://www.elixirconf.eu/talks/dont-let-it-crash-how-we-applied-our-security-checks-on-elixir-code/.
[26] Faculty of Informatics, Eötvös Loránd University, The refactorerl project, 2025. URL: https://plc.inf.elte.hu/erlang/refactorerl-releases.html, accessed: 11-08-2025.
[27] Hex, Github repository, 2025. URL: https://github.com/hexpm/hexpm, accessed: 29-04-2025.
[28] Changelog, Github repository, 2025. URL: https://github.com/thechangelog/changelog.com, accessed: 29-04-2025.
[29] Plausible Analytics, Github repository, 2025. URL: https://github.com/plausible/analytics, accessed: 29-04-2025.
[30] Authex, Github repository, 2025. URL: https://github.com/tino415/authex, accessed: 29-04-2025.