SUBGRAPH

FINDINGS REPORT

SOAP

January 17, 2020

Prepared for: OpenTechFund Red Team Labs / Gem Barrett

Subgraph Technologies, Inc.
642 Rue de Courcelle, Suite 309
Montreal, Quebec
https://subgraph.com

Overview

The OpenTechFund's Red Team Labs was contracted to perform a rapid assessment of the alpha version of SOAP. SOAP is a web-based application for the guided creation of a security policy document specifically tailored for NGOs. SOAP provides a basic document structure, boilerplate content, and an interactive workflow designed for a non-technical user. The output is a document in several formats that can be shared and/or hosted elsewhere.

The objective of the assessment is to understand the threat model and identify security vulnerabilities that could introduce risk to any organization using it.

Summary and Observations

SOAP is an in-browser (client-only) HTML/CSS app that has minimal *natural* vectors for attack due to its size and design.

The application serves as a very lightweight content creation tool that guides a user through creation of a structured organizational security policy. The workflow is one-way, i.e., cannot be traversed backwards through navigational links, and has no persistence: once policy creation begins, it cannot be saved and resumed later. Created policies cannot be edited using the tool once they're finished. If they are not saved at the time the user finishes the workflow, they must be restarted.

The output object is produced entirely by Javascript code within the browser at runtime. If the browser tab or window is closed, or moved, the workflow must be restarted.

Obviously policy changes will occur, but the user would need to edit the output products (text, HMTL, markdown) outside of the SOAP tool or create a derived policy from scratch that includes the desired changes.

Application Attack Surface

Due to the design, there is no apparent way to enter a step in the workflow that isn't the first step such that outside parameters can be injected. Once the user begins creating a policy, they are in complete control until the process is complete or abandoned. Additionally, there are also no outside dependencies.

These properties greatly reduce the attack surface exposed to attackers. Users *can* inject HTML, etc, into input fields, and that content is not filtered, but the result is that they get a document that they themselves created knowingly: including markup, script code, etc. In an attack scenario against a web application, it is almost always the case that there exists a method for an attacker to inject such content without the user being aware, and contrary to their intention.

It could be argued that there is risk that users incorporate hostile content themselves unknowingly via e.g. copy/paste, which then ends up hosted in an organization website, but this is a highly contrived scenario and there are an infinite number of ways this could occur that don't involve SOAP.

Deployment Considerations

SOAP can be used online at: https://usesoap.app.

The incentives for self-hosting may be:

- Customization of template content
- Local Internet access constraints
- Trust/security

SOAP provides a simple Python script to start a local HTTP server. This is intended to be used for testing.

There is no documentation on deployment yet, but it is assumed that users will be instructed to host the static files that comprise SOAP at a location with HTTPS.

Conclusion and Recommendations

Subgraph has concluded that at this stage of the project's existence there are no meaningful attack vectors that could introduce substantial risk with the use of SOAP.

However, the project is considered alpha, and future enhancements will introduce complexity and risk. We imagine features that allow for saving partially completed work, either in a server data store or using browser local storage, and perhaps a means for multiple individuals to collaborate. We recommend a re-visit of SOAP at a point at which sufficient complexity has been introduced.

Subgraph also recommends some documentation guidance on the following:

1) Deployment: the python SimpleHTTPServer does not offer TLS. Users who wish to self-host should be pointed to object stores as the safest way, as this eliminates the need to manage a server. Amazon S3 would be a safe way to host this application. Subgraph recommends that documentation be produced that encourages the user to choose such an option, with step-by-step instructions for doing so. GitHub Pages is another, similar option.

2) A small note about the lack of content filtering. Any HTML, Javascript, etc included in content will pass right through into the finished product. At this point it is only a concern if the user themselves puts such content in, which they may do without being aware via copy/paste. As mentioned above, this is not really a valid attack scenario in our opinion. However, content filtering will need to be introduced if the application becomes more complex and the existence of a dynamic application state or persistence permits attacks such as DOM XSS. Subgraph recommends that robust frameworks be used to manage this, as these will already include protections for these kinds of attacks.

Appendix

Methodology

Our approach to testing is designed to understand the design, behavior, and security considerations of the assets being tested. This helps us to achieve the best coverage over the duration of the test.

To accomplish this, Subgraph employs automated, manual and custom testing methods. We conduct our automated tests using the industry standard security tools. This may include using multiple tools to test for the same types of issues. We perform manual tests in cases where the automated tools are not adequate or reveal behavior that must be tested manually. Where required, we also develop custom tools to perform tests or reproduce test findings.

The goals of our testing methodology are to:

- Understand the expected behavior and business logic of the assets being tested
- Map out the attack surface

- Understand how authentication, authorization, and other security controls are implemented
- Test for flaws in the security controls based on our understanding
- Test every point of input against a large number of variables and observe the resulting behavior
- Reproduce and re-test findings
- Gather enough supporting information about findings to enable us to classify, report, and suggest remediations

---

Description of testing activities

Depending on the type and scope of the engagement, our methodology may include any of the following testing activities:

1. **Information Gathering:** Information will be gathered from publicly availble sources to help increase the success of attacks or discover new vulnerabilities
2. **Network discovery:** The networks in scope will be scanned for active, reachable hosts that could be vulnerable to compromise
3. **Host Vulnerability Assessment:** Hosts applications and services will be assessed for known or possible vulnerabilities
4. **Application Exploration:** The application will be explored using manual and automated methods to better understand the attack surface and expected behavior
5. **Session Management:** Session management in web applications will be tested for security flaws that may allow unauthorized access
6. **Authentication System Review:** The authentication system will be reviewed to determine if it can be bypassed
7. **Privilege Escalation:** Privilege escalation checks will be performed to determine if it is possible for an authenticated user to gain access to the privileges assigned to another role or administrator
8. **Input Validation:** Input validation tests will be performed on all endpoints and fields within scope, including tests for injection vulnerabilities (SQL injection, cross-site scripting, command injection, etc.)
9. **Business Logic Review:** Business logic will be reviewed, including attempts to subvert the intended design to cause unexpected behavior or bypass security controls

---

Reporting

Findings reports are peer-reviewed within Subgraph to produce the highest quality findings. The report includes an itemized list of findings, classified by their severity and remediation status.

Severity ratings

Severity ratings are a metric to help organizations prioritize security findings. The severity ratings we provide are simple by design so that at a high-level they can be understood by different audiences. In lieu of a complex rating system, we quantify the various factors and considerations in the body of the security

findings. For example, if there are mitigating factors that would reduce the severity of a vulnerability, the finding will include a description of those mitigations and our reasoning for adjusting the rating.

At an organization's request, we will also provide third-party ratings and classifications. For example, we can analyze the findings to produce *Common Vulnerability Scoring System* (CVSS)[1] scores or *OWASP Top 10*[2] classifications.

The following is a list of the severity ratings we use with some example impacts:

Critical

Exploitation could compromise hosts or highly sensitive information

High

Exploitation could compromise the application or moderately sensitive information

Medium

Exploitation compromises multiple security properties (confidentiality, integrity, or availability)

Low

Exploitation compromises a single security property (confidentiality, integrity, or availability)

Informational

Finding does not pose a security risk or represents suspicious behavior that merits further investigation

The severity of a finding is often a product of the impact to general security properties of an application, host, network, or other information system.

These are:

Compromise of confidientiality	Exploitation results in authorized access to data
Compromise of integrity	Exploitation results in the unauthorized modification of data or state
Compromise of availability	Exploitation results in a degradation of performance or an inability to access resources

[1] https://www.first.org/cvss/
[2] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

The actual severity of a finding may be higher or lower depending on a number of other factors that may mitigate or exacerbate it. These include the context of the finding in relation to the organization as well as the likelihood of exploitation. These are described in further detail below.

Contextual factors

Confidentiality, integrity, and availability are one dimension of the potential risk of a security finding. In some cases, we must also consider contextual factors that are unique to the organization and the assets tested.

The following is a list of those factors:

Financial compromise	Exploitation may result in financial losses
Reputation compromise	Exploitation may result in damage to the reputation of the organization
Regulatory liability	Exploitation may expose the organization to regulatory liability (e.g. place them in a state of non-compliance)
Organizational impact	Exploitation may disrupt the operations of the organization

Likelihood

Likelihood measures how probable it is that an attacker exploit a finding.

This is determined by numerous factors, the most influential of which are listed below:

Authentication required	Whether or not the attack must be authenticated
Privilege level	Whether or not an authenticated attacker requires special privileges
Public exploit	Whether or not exploit code is publicly available
Public knowledge	Whether or not the finding is publicly known
Exploit complexity	How complex it is for a skilled attacker to exploit the finding
Local vs. remote	Whether or not the finding is exposed to the network
Accessibility	Whether or not the affected asset is exposed on the public Internet
Discoverability	How easy it is for the finding to be discovered by an attacker
Dependencies	Whether or not exploitation is dependant on other findings such as information leaks

Remediation status

As part of our reporting, remediation recommendations are provided to the client. To help track the issues, we also provide a remediation status rating in the findings report.

In some cases, the organization may be confident to remediate the issue and test it internally. In other cases, Subgraph works with the organization to re-test the findings, resulting in a subsequent report reflecting remediation status updates.

If requested to re-test findings, we determine the remediation status based on our ability to reproduce the finding. This is based on our understanding of the finding and our awareness of potential variants at that time. To reproduce the results, the re-test environment should be as close to the original test environment as possible.

Security findings are often due to unexpected or unanticipated behavior that is not always understood by the testers or the developers. Therefore, it is possible that a finding or variations of the finding may still be present even if it is not reproducible during a re-test. While we will do our best to work with the organization to avoid this, it is still possible.

The findings report includes the following remediation status information:

Resolved
Finding is believed to be remediated, we can no longer reproduce it

In progress
Finding is in the process of being remediated

Unresolved
Finding is not remediated – may indicate the initial report, a finding under investigation, or one that the organization has chosen not to address

Not applicable
There is nothing to resolve, this may be the case with informational findings