Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Security-Review Report node_exporter 07.2020

Cure53, Dr.-Ing. M. Heiderich, MSc. D. Weißer, Dr. N. Kobeissi

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

*“Prometheus exporter for hardware and OS metrics exposed by *NIX kernels, written in Go with pluggable metric collectors.”*

From https://github.com/prometheus/node_exporter

This report documents the findings of a security assessment targeting the Prometheus node_exporter complex. Cure53 carried out this examination in summer 2020, revealing six security-relevant issues on the scope. Enveloping a penetration test and an audit of the security posture, the project notably focused not only on the node_exporter software, but also its corresponding codebase and the relevant cryptographic components.

To give some context, the pentest and audit was requested by the maintainers of the Prometheus project, yet the budget was granted by CNCF. The core testing work took place in late June and early July 2020, precisely in CW27 and CW28. Given the nature of the project and the usual process followed by Cure53 when working with CNCF, the chosen methodology was white-box. All relevant code is available as open source and Cure53 also got access to a demo instance integrating the software.

As for the resources, the test and auditing were done by a team of three Cure53 consultants who spent a total of ten days on the project, as requested by the Prometheus team. To best address the objectives of this engagement, the work was split into several smaller work packages (WPs). Within the established structure, WP1 entailed a code audit of the node_exporter, covering all parts deemed as audit-worthy. In WP2, Cure53 examined the general posture of the node_exporter, with the emphasis on code and project. Rounding up the scope, WP3 tackled the cryptographic premise connected to the TLS parts within the node_exporter complex.

As can be seen, a major part of the test and auditing focus was directed to the codebase and the cryptography- or TLS-related parts contained in these. The remaining effort went into a security posture audit seen from a meta-level perspective. This included a high-level look at the project’s closer perimeter, security response mechanisms and other high-level characteristics.

The project started on time and progressed efficiently. The communications during this project were done using a dedicated and private Slack channel created on the Cure53’s workspace. Relevant personnel from the Prometheus and node_exporter teams were invited to join the discussions there. At the same time, not much communication was needed, the scope was absolutely clear and Cure53 made good progress swiftly. Very good coverage levels were reached. Over the course of the assessment, Cure53

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

frequently updated the Prometheus team about the findings by sending over the headlines.

Moving on to the findings, the aforementioned total of six issues were spotted. Two were classified to be security vulnerabilities of varying impact and four represent general weaknesses with lower exploitation potential. No issue reached a *Critical* severity during tests and audits, though the severity levels given to the spotted problems range from *High* to *Low.* One of the most pressing problems, for instance, marks an XSS vulnerability which can only be exploited on MSIE while other browsers are not affected by it to the best of Cure53's knowledge.

In the following sections, the report will first shed light on the scope and key test parameters of this exercise. After that, a dedicated chapter will elaborate on the chosen test methodology and coverage. Next, all findings will be discussed in a chronological order alongside technical descriptions, as well as PoC and mitigation advice when applicable. Finally, the report will close with broader conclusions about this 2020 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team's observations and collected evidence. Tailored hardening recommendations and meta-level advice concerning improvements that could prove valuable for the Prometheus node_exporter project are also incorporated into the final section.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Penetration Test & Code Audit against Prometheus node_exporter Tool**
  - **WP1**: node_exporter Code audit, covering the audit-worthy parts
    - https://github.com/prometheus/node_exporter/tree/release-1.0
  - **WP2**: node_exporter General Posture Audit, covering Code & Project
  - **WP3**: node_exporter Crypto audit for the TLS parts
- **A Test-System was provided for Cure53**
  - https://demo.do.prometheus.io/
  - http://node.demo.do.prometheus.io:9100/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Test Methodology

The following paragraphs describe the metrics and methodologies used to evaluate the security posture of the node_exporter project and codebase. In addition, it includes results for individual areas of the project's security properties that were either selected by Cure53 or singled out by other involved parties as needing closer inspection.

As with previous tests for CNCF, this assignment was also divided into two phases. The general security posture and maturity of the audited code base - here of node_exporter - has been examined in Phase 1. The usage of external frameworks has been audited, security constraints for node_exporter configurations were examined and the documentation has been deeply studied in order to get a general idea of security awareness at the node_exporter complex. This was followed by research into how security reports and vulnerabilities are handled and how seriously the entire standpoint towards a healthy security infrastructure is taken. The latter phase covered actual tests and audits against the node_exporter codebase, with the code quality and its hardening being judged.

Phase 1: General security posture checks

As mentioned earlier, Phase 1 enumerated general qualities of the audited project. Here, a meta-level perspective on the general security posture is reached by providing details about the language specifics, configurational pitfalls and general documentation. An additional view on how node_exporter handles vulnerability reports and how the disclosure process works is provided as well. A perception rooted in the maturity of node_exporter is given, solely from a wider, broad brush strokes perspective. Specific impressions linked to the code quality relate to Phase 2 of the audit process.

Phase 2: Manual code auditing

For this component, Cure53 performed a small-scale code review and attempted to identify security-relevant areas of the project's codebase and inspect them for flaws that are usually present in distributed systems. This is an addition to the previous maturity analysis and supplies a more detailed perspective on the project's implementation when it comes to security-relevant portions of the code. Still, this Phase was limited by the budget and cannot be seen as complete without a large-scale code review with in-depth analysis of the multiple parts forming the project's scope. As such, the goal was not to reach extensive coverage, but to gain an impression about the overall quality of node_exporter and determine which parts of the project's scope deserve thorough audits in the future.

Later chapters in this report will also elaborate on what was being inspected, why and with which implications for the node_exporter metrics collector.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Phase 1: General security posture checks

This Phase is meant to provide a more detailed overview of the node_exporter project's security properties, which are seen as somewhat separate from both the code and the node_exporter software.

The first few subsections of a posture audit focus on more abstract components of a specific project instead of judging the code quality itself. Later subsections look at elements that are linked more strongly to the organizational and team-level aspect of node_exporter. In addition to the items presented below, the Cure53 team also focused on the following tasks to be able to conduct a cross-comparative analysis of all observations.

- The documentation was examined to understand all provided functionality and acquire examples of what a real-world deployment of node_exporter looks like.
- The main control flow of the node_exporter application was followed and the main structure of the codebase has been analyzed.
- High-level code audits have been conducted. This was necessary to get a quick impression of the overall style and to reach an understanding of which areas are interesting for a more deep-dive approach in Phase 2 of the audit.
- Normally, past vulnerability reports in node_exporter would have been checked out to spot interesting areas that suffered in the past. However, node_exporter never received a vulnerability report.
- Concluding on the steps above, the project's maturity was evaluated; specific questions about the software were compiled from a general catalogue according to individual applicability.

Application/Service/Project Specifics

In this section, Cure53 describes the areas which were inspected for having insight into the application-specific aspects that lead to a good security posture. These include choice of programming language, selection and oversight of external third-party libraries, as well as other technical aspects like logging, monitoring, test coverage and access control.

Language Specifics

Programming languages can provide functions that pose an inherent security risk and their use is either deprecated or discouraged. For example, *strcpy()* in C has led to many security issues in the past and should be avoided altogether. Another example would be the manual construction of SQL queries versus the usage of prepared statements. The choice of language and enforcing the usage of proper API functions are therefore crucial for the overall security of the project.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

node_exporter is written in Go, which inherently provides memory safety and broadly offers a higher level of security in comparison to e.g. C/C++. This is further underlined by only making use of the Go's *unsafe* package if absolutely necessary, in particular when obtaining system metrics. While some collectors have native C code embedded, user input does not reach those sinks. The code is written with best practices in mind, helping not only with auditing, but also with maintenance. The above indicators contribute to a healthy security posture and seem well-understood and properly distributed throughout the node_exporter codebase. Specific examples include:

- Nesting being avoided by handling errors first;
- Separating test-cases from code;
- Keeping documentation/items concise;
- Separating independent packages;
- Avoiding unnecessary repetitions.

External Libraries & Frameworks

While external libraries and frameworks can also contain vulnerabilities, it is nonetheless beneficial to rely on sophisticated libraries instead of reinventing the wheel with every project. This is especially true for cryptographic implementations, since those are known to be prone to errors.

As almost every project, node_exporter makes use of external libraries and avoids reimplementing existing solutions. For example, several Prometheus packages are utilized to read information from the *procfs* and to format the output properly. The integrated web server is provided by the *net/http* package. No concerns were found to be present in the used third-party packages in general. All appear to be widely recognized by the community and to be under active development.

Configuration Concerns

Complex and adaptable software systems usually have many variable options which can be configured accordingly to the actually deployed application requirements. While this is a very flexible approach, it also leaves immense room for mistakes. As such, it often creates the need for additional and detailed documentation, in particular when it comes to security.

More to the point, node_exporter allows to enable and disable the individual collectors via the configuration, letting users choose which metrics are being exposed. Additionally, it is possible to configure TLS certificates and to set up mandatory user credentials.

In a default configuration, a subset of the collectors is enabled but no further mitigations are active. This comes due to the fact that SSL certificates and passwords cannot be

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

shipped pre-configured and must be generated individually by the user. However, the benefits of having this additional layer of security should be more visible in the project's documentation.

The included startup scripts execute the daemon as a dedicated user where possible, which has been properly adopted by widely used Linux distros like *Archlinux* and *Debian*. This further limits the risk for privilege escalations as local attackers would not gain anything by targeting the node_exporter.

Access Control

Whenever an application needs to perform a privileged action, it is crucial that an access control model is in place, to ensure that appropriate permissions are present. Furthermore, if the application provides an external interface for interaction purposes, some form of separation and individual access control may be required.

Obtainable system metrics can be specified via the configuration file and extra parameters on startup. Furthermore, it is possible to prevent unauthorized access by adding user credentials to the configuration. User-based permissions cannot be configured and credentials will always give access to all enabled metrics, which is not necessarily a bad thing for a small application like node_exporter. If demand for a more fine-grained configuration arises, it can still be implemented.

Logging/Monitoring

Having a good logging/monitoring system in place allows developers and users to identify potential issues more easily, or get a rough idea of what is going wrong. It can also provide security-relevant information, for example when a verification of a signature fails. Consequently, having such a system in place has a positive influence on the project.

In this realm, node_exporter has an integrated logger which has four different verbosity levels ranging from *Debug* to *Error*. While a configuration set to *Error* only monitored events are logged, *Debug* logs every execution of the collectors. However, it is not possible to log client IP addresses, and even failed authentications can go unnoticed. For administrators who need to monitor the service itself, such metrics would be important in order to detect potential attacks.

Unit/Regression and Fuzz-Testing

While tests are essential for any project, their importance grows with the scale of the endeavor. Especially for large-scale compounds, testing ensures that functionality is not broken by code changes. Furthermore, it generally facilitates the premise where features function the way they are supposed to. Regression tests also help guarantee that

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

previously disclosed vulnerabilities do not get reintroduced into the codebase. Testing is therefore essential for the overall security of the project.

Test cases cover the core functionality of node_exporter, including the web server and the TLS configuration in addition to a few collectors. While the most important aspects are tested, it cannot be said that full coverage is achieved, leaving room to improve in terms of testing.

The application is written in a language that is generally memory safe and does almost no parsing of contents an unprivileged user can control. These two factors eliminate the need for fuzzing.

Documentation

Good documentation contributes greatly to the overall state of the project. It can ease the workflow and ensure final quality of the code. Having a coding guideline which is strictly enforced during the patch review process for example, ensures that the code is readable and can be easily understood by a spectrum of developers. Following good conventions can also reduce the risk of introducing bugs and vulnerabilities to the code.

There are two different documentation pages for the node_exporter project. While the documentation hosted on the Prometheus project page[1] focuses on installation and usage, the *README*[2] mainly describes the collectors. Notes on securing the service are not part of the main *README* and easy to overlook if not explicitly searched for. As a secure standard configuration is infeasible in this context, the documentation should state clearly that the node_exporter service is not protected by default and can be used by anyone with network access.

Organization/Team/Infrastructure Specifics

This section will describe the areas Cure53 looked at to find out about the security qualities of the node_exporter project which cannot be linked to the code and software but rather encompass handling of incidents. As such, it tackles the level of preparedness for critical bug reports within the node_exporter development team.

In addition, Cure53 also investigated the degree of community involvement, i.e. through the use of bug bounty programs. While a good level of code quality is paramount for a good security posture, the processes and implementations around it can also make a difference in the final assessment of the security posture.

[1] https://prometheus.io/docs/guides/node-exporter/
[2] https://github.com/prometheus/node_exporter/blob/master/README.md

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Security Contact

To ensure a secure and responsible disclosure of security vulnerabilities, it is important to have a dedicated point of contact. This person/team should be known, meaning that all necessary information such as an email address and preferably also encryption keys of that contact should be communicated appropriately.

The *MAINTAINERS*[3] file lists email addresses of project maintainers that can be contacted to report vulnerabilities, but there is no specific guideline on how to report security issues in node_exporter. Furthermore, the document omits important details, such as the respective PGP keys and an outline of the disclosure process. Although the PGP keys can be obtained from public key servers it is advised to link them in the document. It is recommended to put all details related to reporting and disclosing security issues in a dedicated *SECURITY file* in the project's repository.

Security Fix Handling

When fixing vulnerabilities in a public repository, it should not be obvious that a particular commit addresses a security issue. Moreover, the commit message should not give a detailed explanation of the issue. This would allow an attacker to construct an exploit based on the patch and the provided commit message prior to the public disclosure of the vulnerability. This means that there is a window of opportunity for attackers between public disclosure and widespread patching or updating of vulnerable systems. Additionally, as part of the public disclosure process, a system should be in place to notify users about fixed vulnerabilities.

At this point in time, it cannot be evaluated how security fixes are handled and how they are disclosed. This is because there are no public vulnerability reports, no CVEs and none of the commits mentions that a security issue was fixed.

Bug Bounty

Having a bug bounty program acts as a great incentive in rewarding researchers and getting them interested in projects. Especially for large and complex projects that require a lot of time to get familiar with the codebase, bug bounties work on the basis of the potential reward for efforts.

The node_exporter project does not have a bug bounty program at present, however this should not be strictly viewed in a negative way. This is because bug bounty programs require additional resources and management, which are not always a given for all projects. However, if resources become available, establishing a bug bounty program for

[3] https://github.com/prometheus/node_exporter/blob/master/MAINTAINERS.md

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

node_exporter should be considered. It is believed that such a program could provide a lot of value to the project.

Bug Tracking & Review Process

A system for tracking bug reports or issues is essential for prioritizing and delegating work. Additionally, having a review process ensures that no unintentional code, possibly malicious, is introduced into the codebase. This makes good tracking and review into two core characteristics of a healthy codebase.

Bug reports are handled via the GitHub issue tracker but no exhaustive guideline on how to file tickets exists. There is however a GitHub specific issue template which states what to include in a report.

Users can submit small fixes to the *node_exporter* project via pull requests on GitHub but larger contributions should be discussed on the mailing list first. The corresponding workflow is explained in the project's *CONTRIBUTING* file, which is considered suitable for open source projects. Contributions are reviewed by node_exporter maintainers in order to prevent the submission of malicious or dysfunctional code.

Evaluating the Overall Posture

The overall security posture of the node_exporter project still has some room to improve but there is no reason to be concerned as the related items are easy to address. By improving the documentation in terms of security, adding details for security reports and expanding the logging of the service itself, the most important flaws can be resolved. A few parts of the posture audit were found inapplicable. How node_exporter handles vulnerability reports and disclosure processes for example, will remain to be seen in the future.

Choosing Go has been a great decision and automatically reduces the potential for introducing memory safety-related issues. The fact that almost no user input from remote sources is parsed, and the small attack surface in general, makes it difficult for potential intruders to run meaningful attacks. Furthermore, running the service as a dedicated user eliminates the risk of potential local privilege escalations.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Phase 2: Manual code auditing & pentesting

This section comments on the code auditing coverage within areas of special interest and documents the steps undertaken during the second phase of the audit against the node_exporter software complex.

- node_exporter was audited from multiple angles, so that different attack models could be covered including local and remote adversaries.
- One of the developer's main concerns was the TLS implementation and if any leaks can occur. This part was thoroughly audited in order to find potential flaws that could lead to man-in-the-middle attacks.
- The code that handles the HTTP Basic authentication was examined to ensure that no obvious bypasses are possible as well.
- Further investigations targeted the server itself, aiming to find flaws which could lead to information leakage or even a takeover of the system. Since there is basically only one parameter that can be remotely provided, there is not much potential for remote attacks.
- The possibility of local privilege escalations was checked, but as node_exporter suggests running the service as a dedicated user and modern Linux distributions follow suit, there is very little an attacker could gain from such an attempt.
- The node_exporter code was also inspected for stability and if the failure of a single collector can render the entire application useless.
- Some of the collectors embed native C code for direct interactions with the operating system. The relevant sections were checked for potential memory safety issues.
- The status server component was checked in regard to the exposed HTTP endpoints. One of two debug endpoints (*/debug/pprof/hea*p) could potentially lead to leaking sensitive heap information.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Every vulnerability is additionally given a unique identifier (e.g. *PRM-02-001*) for the purpose of facilitating any future follow-up correspondence.

PRM-02-002 Web: Reflected XSS on MSIE due to unsanitized parameter *(Low)*

It was discovered that the *collect* parameter is directly reflected inside of an error message in case the given collector does not exist. All responses from the web server have the content type set to *text/plain* making exploitation of XSS vulnerabilities much more difficult as modern web browsers will not evaluate HTML in this context. There exists an edge case however, where *MSIE11* on Windows 7 still renders HTML even with a *text/plain* content type. This is described in more detail in the Cure53's *Browser Security White Paper*[4]:

> *"The first edge case here is a frame redirect working on MSIE11. It is possible to cause XSS from within an application/json response by loading it in an iframe that uses a very fast navigation pattern. This approach would confuse MSIE11 about the actual Content-Type - which is benign JSON in this case - and have it rendered as HTML instead."*

Given how difficult this issue is to actually exploit, and how minor the impact of an XSS attack for this case would be, the severity of the flaw has been set to *Low.* A sample payload for exploiting the issue is shown below.

***poc.eml*:**

```
<meta http-equiv="X-UA-Compatible" content="IE=5">
<iframe src="http://target:9100/metrics?collect[]=AAA<?
PXML><html:script>alert(1)</
html:script>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAA"></iframe>
```

[4] https://github.com/cure53/browser-sec-whitepaper

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

It is recommended to set *X-Frame-Options*, *X-Content-Type-Options* and other security-relevant headers as pointed out in PRM-02-001. More importantly, the HTML characters must be encoded in the response.

PRM-02-005 TLS: Insecure TLS versions accepted *(High)*

It was observed that node_exporter mandates a user-provided TLS configuration in order to enable transport security, and that the aforementioned configuration accepts values which allow TLS variants known to be insecure, such as TLS 1.0 and TLS 1.1.

TLS 1.0 and TLS 1.1 are known to be broken[5], having recently been removed entirely from Mozilla Firefox[6] and Google Chrome[7]. Accepting user configurations for TLS 1.0 and TLS 1.1 constitutes an unnecessary security risk.

Affected File:
https/tls_config.go

Affected Code:

```
func Listen(server *http.Server, tlsConfigPath string, logger log.Logger) error
{
	if tlsConfigPath == "" {
		level.Info(logger).Log("msg", "TLS is disabled and it
			cannot be enabled on the fly.", "http2", false)
		return server.ListenAndServe()
	}
	if err := validateUsers(tlsConfigPath); err != nil {
		return err
	}
	// Setup basic authentication.
	var handler http.Handler = http.DefaultServeMux
	if server.Handler != nil {
		handler = server.Handler
	}
	server.Handler = &userAuthRoundtrip{
		tlsConfigPath: tlsConfigPath,
		logger:        logger,
		handler:       handler,
	}
	c, err := getConfig(tlsConfigPath)
[...]
func getConfig(configPath string) (*Config, error) {
	content, err := ioutil.ReadFile(configPath)
	if err != nil {
```

[5] https://www.comodo.com/e-commerce/ssl-certificates/tls-1-deprecation.php
[6] https://hacks.mozilla.org/2020/02/its-the-boot-for-tls-1-0-and-tls-1-1/
[7] https://www.chromestatus.com/feature/5759116003770368

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
                return nil, err
        }
        c := &Config{
                TLSConfig: TLSStruct{
                        MinVersion:               tls.VersionTLS12,
                        MaxVersion:               tls.VersionTLS13,
                        PreferServerCipherSuites: true,
                },
                HTTPConfig: HTTPStruct{HTTP2: true},
        }
        err = yaml.UnmarshalStrict(content, c)
        return c, err
}
```

It is recommended that users be prohibited from specifying minimum and maximum TLS versions at all, and the default value of TLS 1.2 being the minimum version and TLS 1.3 being the maximum version to be constantly adopted across all transport-secured connections and configurations.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

PRM-02-001 Web: General HTTP security headers missing *(Medium)*

node_exporter is not meant to run as a standard web application and the content type being set to text/plain renders lots of attacks infeasible. However, as the HTTP protocol is used, many web-based attacks still apply which is why some standard countermeasures should be implemented.

It was found that the node_exporter web server is missing certain HTTP security headers in HTTP responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other problems. The following list enumerates the headers that need to be reviewed to prevent this and similar flaws.

- **X-Frame-Options**: This header specifies whether the web page is allowed to be framed. Although this header is known to prevent Clickjacking attacks, there are many other attacks which can be achieved when a web page is frameable[8]. It is recommended to set the value to either SAMEORIGIN or DENY.

[8] https://cure53.de/xfo-clickjacking.pdf

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

- Note that the CSP framework offers similar protection to X-Frame-Options in ways that overcome some of the shortcomings of the aforementioned header. To optimally protect users of older as well as modern browsers at the same time, it is recommended to consider deploying the *Content-Security-Policy: frame-ancestors 'none';* header as well.
- **X-Content-Type-Options**: This header determines whether the browser should perform MIME Sniffing on the resource. The most common attack abusing the lack of this header is tricking the browser to render a resource as an HTML document, effectively leading to Cross-Site-Scripting (XSS).
- **X-XSS-Protection**: This header specifies if the browser's built-in XSS auditors should be activated (enabled by default). Not only does setting this header prevent Reflected XSS, but also helps to avoid the attacks abusing the issues on the XSS auditor itself with false positives, e.g. Universal XSS[9] and similar. It is recommended to set the value to either 0 or 1; mode=block. Note that most modern browsers have stopped supporting XSS filters in general, so this header is only relevant in case older browsers are supported by the web application in scope.
- **Strict-Transport-Security**: Without the HSTS header, a MitM attacker could attempt to perform channel downgrade attacks using readily available tools such as *sslstrip*[10]. In this scenario the attacker would proxy clear-text traffic to the victim-user and establish an SSL connection with the targeted website, stripping all cookie security flags if needed. It is recommended to set up the header as follows:

  Strict-Transport-Security: max-age=31536000; includeSubDomains;

  Note: the HSTS *preload* flag has been left out as it is considered dangerous[11].

Overall, missing security headers is a bad practice that should be avoided. It is recommended to add the following headers to every server response, including error responses like *4xx* items. More broadly, it is recommended to reiterate the importance of having all HTTP headers set at a specific, shared and central place rather than setting them randomly. This should either be handled by a load balancing server or a similar infrastructure. If the latter is not possible, mitigation can be achieved by using the web server configuration and a matching module.

[9] http://www.slideshare.net/masatokinugawa/xxn-en
[10] https://moxie.org/software/sslstrip/
[11] https://www.tunetheweb.com/blog/dangerous-web-security-features/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

PRM-02-003 Collectors: Denial-of-Service in *textfile* collector via *FIFO* file *(Low)*

In order to gather metrics that are not covered by node_exporter itself, external applications can provide data via *text* files placed in a designated directory.

As the service blindly reads all files with a *.prom* extension, this feature can be abused to cause a Denial-of-Service. A local user with write permissions to the folder, where external metrics are fetched from, can create a *FIFO* file which blocks on reading and renders the node_exporter service useless. As this already requires an attacker with relatively high privileges and due to the fact that a non-responsive metrics service could increase the attacker's risk of being detected, the overall impact of the issue is fairly limited.

It is recommended to consider blocking special files like pipes completely or to implement a timeout that prevents blocking *FIFO* devices from causing a Denial-of-Service.

PRM-02-004 Collectors: DoS in *supervisord* collector via invalid response *(Low)*

Metrics from *supervisord* can be fetched by providing node_exporter with an RFC-URL, which is expected to return data in an XML format. It was discovered that a malicious server can return data which crashes the node_exporter service due to an unexpected data-type. Shown below is a response that contains an integer instead of an array leading to this error:

RFC Response:

```
<methodResponse>
  <params>
    <param>
      <value>
        <int>1</int>
      </value>
    </param>
  </params>
</methodResponse>
```

Crash Log:

```
panic: interface conversion: interface {} is int, not xmlrpc.Array

goroutine 91 [running]:
github.com/prometheus/node_exporter/collector.
(*supervisordCollector).Update(0xc000413590, 0xc000429500, 0x56148983e560, 0x0)
       github.com/prometheus/node_exporter/collector/supervisord.go:140 +0xd2f
github.com/prometheus/node_exporter/collector.execute(0x5614890dc790, 0xb,
0x56148936d4a0, 0xc000413590, 0xc000429500, 0x56148936cc80, 0xc000033020)
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
        github.com/prometheus/node_exporter/collector/collector.go:153 +0x86
github.com/prometheus/node_exporter/
collector.NodeCollector.Collect.func1(0xc000429500, 0xc0004134d0,
0x56148936cc80, 0xc000033020, 0xc000492b30, 0x5614890dc790, 0xb, 0x56148936d4a0,
0xc000413590)
        github.com/prometheus/node_exporter/collector/collector.go:144 +0x6f
created by github.com/prometheus/node_exporter/collector.NodeCollector.Collect
        github.com/prometheus/node_exporter/collector/collector.go:143 +0x135
```

It is recommended to implement proper error handling for invalid RFC responses instead of letting the application crash.

PRM-02-006 Web: Runtime profiling data exposed via *pprof* *(Low)*

It was discovered that the *net/http/pprof* package is included in the application, which gives access to extensive debugging information like traces and memory mappings. While this is not a large problem on its own, it could be leveraged favorably in combination with other, more severe vulnerabilities.

Accessing *pprof* profiler:
http://localhost:9100/debug/pprof/

It is recommended to disable the *profiler* in order to prevent the information leak. If there are cases where a process *profiler* is actually required, it is recommended to give users the option to disable the component via the configuration.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

The results of this CNCF-funded assessment of the Prometheus node_exporter complex point to both strengths and weaknesses of the test-targets. After spending ten days on the scope in the summer of 2020, three members of the Cure53 team were unable to spot a *Critical*-scoring problem, yet unveiled issues ranking as *High.*

To give some detailed feedback on the security posture of the node_exporter software and its surroundings, it should be noted that the project looked at both the overall security standing of the complex and the application code. While flaws were found in both areas, no major risks transpired. The Cure53's evaluation of the general security posture revealed that there is room to improve regarding security contacts and reporting of security issues. Further, the insecure-by-default aspect should be explicitly stated in the main documentation. Apart from these minor issues the overall posture made a good impression given the small size of the project.

As argued in Logging/Monitoring, it is recommended to improve the logging mechanism to include requests and especially failed authentication attempts. This would facilitate the detection of intruders. Further, one of the key focus areas were potential information leaks via Man-in-the-Middle attacks which is why a lot of attention was paid to the TLS handling of the project. In particular, the TLS component was fully reviewed. While the configuration component was found to allow insecure variants of TLS (PRM-02-005), no other issues were spotted.

It needs to be underscored that node_exporter relies on the HTTP protocol in order to make the collected information available. Therefore, the application was tested for common web vulnerabilities such as XSS. Despite the service not being meant as a web application and the response content-type being set to *text/plain,* standard web-based flaws cannot be automatically excluded. As addressed in PRM-02-001, missing security headers can still cause problems that allow for information leakage, e.g. via Clickjacking. In a setup with Windows 7 and MSIE11, even XSS is possible, as described in PRM-02-002.

Another important aspect for a service that accepts network connections is the robustness against attacks via parameters the user can specify. Since there is only one parameter that can be controlled remotely (*collect*), there is almost no potential for such flaws. Further tests were conducted with local attackers in mind. As the service is supposed to run as a non-*root* user, privilege escalation flaws would be relatively uninteresting. No issues were found to allow gaining the privileges of the user capable of executing the service.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

From a stability perspective, two issues were found that could render the service dysfunctional. Issue PRM-02-003 describes how a local user could block the *collector* under the right conditions and PRM-02-004 shows how a rogue *supervisord* server crashes node_collector completely. These should be addressed in due course.

All in all, it can be said that there are quite a few issues with both the general posture and the project's source code examined by Cure53 during this July 2020 project. Simultaneously, it is crucial that the discovered flaws are easy to address. Most importantly, no structural failures were spotted during the audit of node_exporter.

Cure53 would like to thank Ben Kochie, and Richard Hartmann from the Prometheus team as well as Chris Aniszczyk of The Linux Foundation, for their excellent project coordination, support and assistance, both before and during this assignment. Special gratitude also needs to be extended to The Linux Foundation for sponsoring this project.