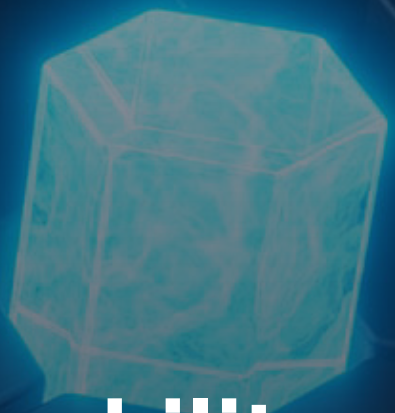
DEFENCE TECH
TINEXTA GROUP



Open source vulnerability hunting: iTop

Vulnerability Analysis Report

Summary

DEFENCE TECH
TINEXTA GROUP

1

Our Malware Lab

1. Our Malware Lab

Defence Tech Malware Lab daily performs dissection of malware with the aim of timely understanding the technological evolutions of attacks, consolidating the knowledge of necessary to make more effective and faster the process of incidents responding, contributing to spreading information about emerging threats into the expert's community and among its clients.

Malware Lab analysts are continuously engaged in searching and experimenting new analysis tools, for increasing accuracy and scope of action with regard to the proliferation of new evasion and anti-analysis techniques adopted by malwares.

The Malware Lab is also committed to the development of proprietary tools for malware analysis and supporting the management and response of incidents.

Besides malware analysis, Malware Lab ideated and implemented an automatic process of extraction of **Indicators of Compromise (IOC)** that is daily run on dozens of new malwares, intercepted in the wide for populating our Knowledge Base.

CORRADO AARON VISAGGIO
Group Chief Scientist Officer & Malware Lab Director
a.visaggio@defencetech.it

DEFENCE TECH
TINEXTA GROUP

2

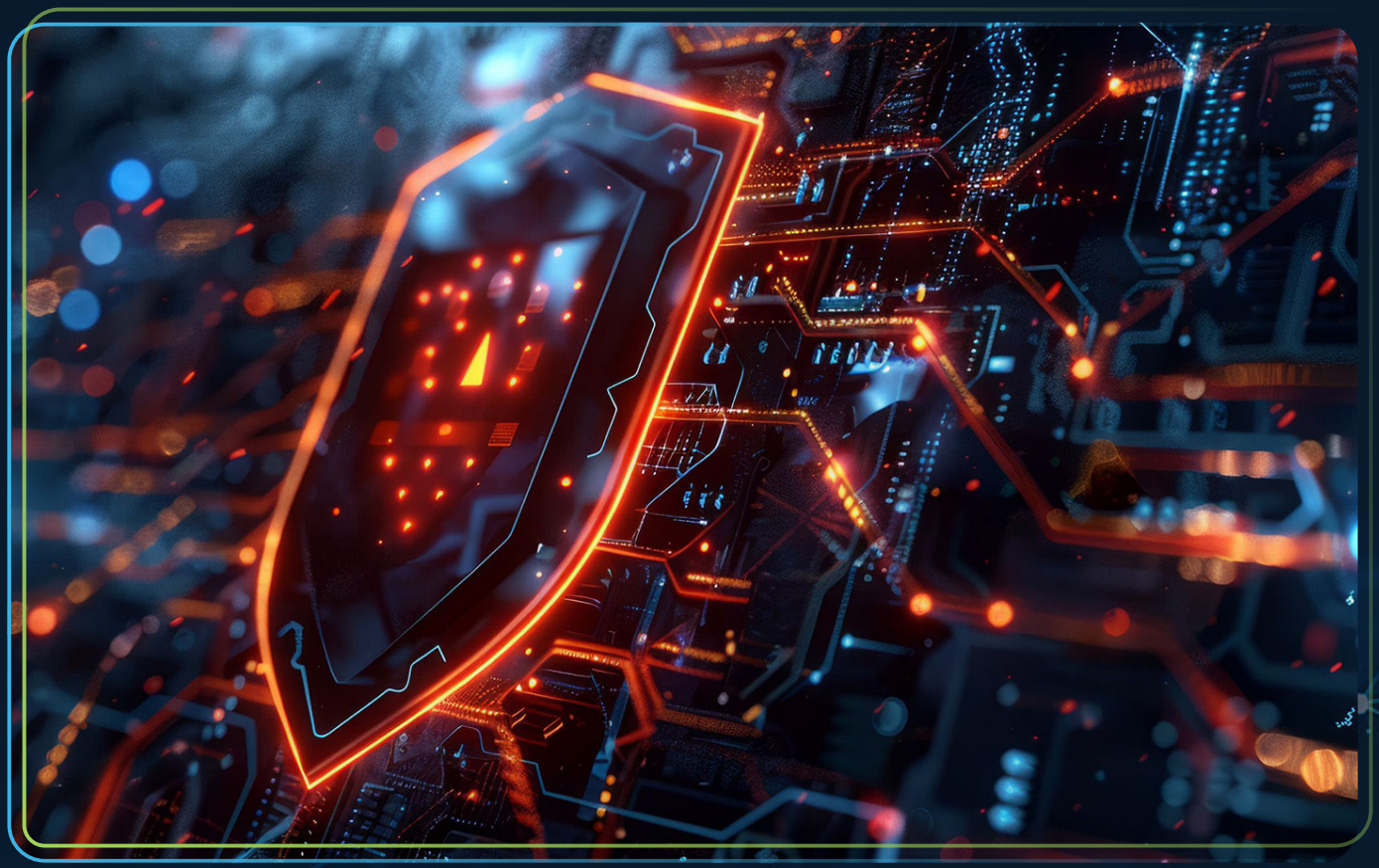
Executive Summary

2. Executive Summary

Before adopting new tools, we usually perform security audits to ensure that we are not introducing a new attack surface within our internal processes. This report is about one such tool we inspected as part of our policy.

iTop[1] is an open source and web-based IT service management platform developed by Combodo[2].

Our team discovered multiple vulnerabilities in its PHP code-base as well as a widespread misconfiguration of online instances. All the issues were privately disclosed to Combodo through their vulnerability report contact and are now being published because either they have been fixed or closed as not eligible for a fix.



[1] *https://github.com/Combodo/iTop*

[2] *https://www.combodo.com/itop-193*

CVE-2024-51995: Logic bug in ajax.render.php allows for bypass of ‘backOffice’ access control

The ajax.render.php page performs access control to prevent portal users, the low-privileged end-user accounts, from performing most actions. In particular we observe that only a handful of API calls are allowed:

```
operation = utils::ReadParam('operation', '', false,
utils::ENUM_SANITIZATION_FILTER_OPERATION);

// Only allow export functions to portal users
switch ($operation) {
    case 'export_build_portal':
    case 'export_cancel':
    case 'export_download':
    case 'cke_img_upload':
    case 'cke_upload_and_browse':
    case 'cke_browse':
        $sRequestedPortalId = null; // Allowed for all users
        break;

    default:
        $sRequestedPortalId = 'backoffice'; // Allowed only for console users
        break;
}
// Whitelist is enforced here
LoginWebPage::DoLoginEx($sRequestedPortalId, false);
```

Github reference for this code snippet

The logic here is designed to require the user to be logged in as backoffice to perform any operation other than the ones listed in the switch statement.

```
operation = utils::ReadParam('operation', '', false,

$sRoute = utils::ReadParam('route', '', false,
utils::ENUM_SANITIZATION_FILTER_ROUTE);
$oRouter = Router::GetInstance();
if ($oRouter->CanDispatchRoute($sRoute)) {
    $mResponse = $oRouter->DispatchRoute($sRoute);
```

Github reference

This snippet takes the route parameter from the request and dispatches it without any further access control.

This means that we can request any route we want as long as we specify an operation that is allowed. Since routing happens before the operation is processed.

We believe that this is an oversight since the same DispatchRoute pattern in UI.php properly enforces access control by calling DoLogin() early. This works because of its default parameter $bIsAllowedToPortalUsers = false which only allows backoffice users.

To exploit this we need to find valid routes that let us perform actions on the server bypassing access control. We found one such API named ObjectController.

We can perform arbitrary queries by constructing an URL like the following:
http://localhost/itop/pages/ajax.render.php?operation=cke_browse&route=object.get

The operation name doesn't matter and it's only needed to bypass the user identity check, in reality this request will end up invoking Combodo\iTop\Controller\Base\Layout\ObjectController::OperationGet even though the user is not logged in as backoffice.

The get and search endpoints can be used to perform arbitrary OQL queries, while the modify method can edit objects not owned by the user. The impact of this issue is explained in the following section as we need to discuss iTop's OQL and the object model first.

This issue has been fixed in iTop 3.1.2 and 3.2.0[3].

[3] *https://github.com/Combodo/iTop/security/advisories/GHSA-3mxr-8r3j-j2j9*

iTop's object model

ITop defines a customizable object model where end users can define their own classes and they can be transparently used throughout the web interface and API, it also includes many built-in classes.

These classes are protected by the access matrix: a data structure that defines what kind of operations can users perform on the data such as read, write, create and so on. The process for configuring these permissions is explained in the documentation.

For certain classes such as user accounts or tickets there are extra ownership checks implemented in the application logic, such as ensuring users can't see tickets created by other users.

The access matrix is implemented deep within the data access layer and it's very hard to bypass however if we can find arbitrary query primitives such as the previously illustrated vulnerability a rogue user can read arbitrary data as long as the access matrix grants it.

The access matrix is a very strong mitigation against logic bugs, for example it is not possible to use the OperationNew method to create a new user account or OperationModify to gain admin access because normal users are forbidden from modifying user instances by the access matrix.

As a counter example, reading user posts is always allowed by the access matrix since users must be able to read at least their own messages, the checks to prevent an user from spying on others are implemented in the logic layer.

Since none of the methods in the ObjectController class perform ownership checks. This means that the following attacks are possible from the context of any portal user:

- List all Person instances, even from other organizations
- List all UserRequest instances (tickets)
- Modify the content of arbitrary UserRequest instances
- Use blind query techniques to extract information from the database that is not exposed by this API.

To perform these queries we use OQL, iTop's query language. It is very similar to SQL but it is interpreted by the data layer, it does not allow direct access to the database.

Interestingly, this vulnerability is mitigated by a bug in the code. The OperationGet method only returns a subset of the object's properties, while the OperationSearch allows requesting any property through the fields_to_load parameter which is meant to contain a JSON array of strings.

However, there is a bug.

```
$aFieldsToLoad = json_decode(utils::ReadParam('fields_to_load', '[]', false,
utils::ENUM_SANITIZATION_FILTER_STRING));
```

Github reference

The parameter is read with the ENUM_SANITIZATION_FILTER_STRING sanitization which transforms most special characters to HTML escape sequences, causing the JSON to be invalid.

We still managed to recover arbitrary properties using the classic blind query technique of guessing the content of a string character by character. This can be done in OQL using the following syntax: SUBSTR(field_name, index, 1) = 'character'.

We built a proof of concept that extracts the temp_file_path property of BulkExportResult instances. Combining this with a common misconfiguration of iTop that exposes the data directory in the webroot (more in the following sections) we were able to download the bulk export files of other users.

While this API is still available, the bug that allowed unauthenticated users to access it has been fixed, however it remains an interesting primitive in case future similar bugs are discovered.

Broken access control in the SOAP interface

The BasicServices::SearchObjects method does not prevent portal users from accessing it. This allows any registered user to perform arbitrary OQL queries to any object class they have access to through the access matrix. This has the same implications as the previous issue but through a different interface. in this case all the fields of the queried objects are returned.

The REST version of the interface has proper access checks in place using the DoLogin function by setting bIsAllowedToPortalUsers = false

$iRet = LoginWebPage::DoLogin(false, false, LoginWebPage::EXIT_RETURN);

Github reference

Combodo did not assign a CVE to this issue claiming that the SOAP interface is deprecated and will be removed. As of now, even the latest version of iTop is vulnerable to this.

We suggest to mitigate this issue by blocking access to the SOAP interface by locking the following pages:

- /webservices/itop.wsdl.php
- /webservices/soapserver.php

CVE-2024-51994: Self XSS in portal picture upload

Error modals are produced using the .html() method, making them prone to XSS attacks. This can happen every time the server returns an error message containing user content that has not been properly sanitized.

A curious example of this comes from the picture upload form in the portal /pages/exec.php/user?exec_module=itop-portal-base&exec_page=index.php&portal_id=itop-portal where uploading a text file containing <script>alert(1)</script> instead of an image will trigger an XSS due to the image library returning the initial bytes of the file as part of the error message.

The overall impact of this specific case is low, however other errors may be manipulated to increase the impact.

This specific issue is caused by the following code snippet:

```
oModalElem.find('.modal-content .modal-header .modal-title').html(sTitle);
oModalElem.find('.modal-content .modal-body .alert').addClass('alert-
danger').html(sBody);
```

Github reference

This issue has been fixed in iTop 2.7.11, 3.1.2 and 3.2.0[4].

[4] *https://github.com/Combodo/iTop/security/advisories/GHSA-jjph-c25g-5c7g*

CVE-2025-27139: Stored Self XSS in preferences

GetListPageSizeFieldBlock() Concatenates the user preference default_page_size with raw html, this can cause an XSS when the preferences page is opened. To exploit this one simply needs to store the malicious value using the set_pref endpoint.

Github reference

Example request:

```
curl localhost/itop/pages/ajax.render.php?operation=set_pref \
  -H "... cookie here..."  \
  --data "code=default_page_size&value='10\"/> <script>alert(1)</script> <div
id=\"test"
```

Then go to http://localhost/itop/pages/preferences.php to trigger the XSS.

This issue was fixed in iTop versions 2.7.12, 3.1.2 and 3.2.0[5].

[5] *https://github.com/Combodo/iTop/security/advisories/GHSA-c6mg-9537-c8cf*

Dangerous HTTP server defaults

iTop stores application data in subdirectories of the web root, this means they could be accessed through unauthenticated HTTP requests.

To prevent such attacks iTop includes configuration files for IIS and Apache to set proper access permissions to these directories, however only IIS respects the configuration out of the box making iTop instances on linux vulnerable by default.

During the initial setup, the wizard warns about the possibility of this misconfiguration linking the relevant documentation but this is not enforced and easily overlooked.

A google search for "index of" "itop" reveals many misconfigured instances that expose sensitive data at best and whole dumps of the database at worst.

The folders in the web root contain unencrypted full database backups, user uploads, plaintext data exports and log files, depending on the environment even database or LDAP credentials.

Even when directory indexing is disabled, many file paths are fixed or predictable, for example by reading the /log/error.log file one might disclose the full path of the periodic automatic backups that are accessible through the data directory as well.

We ran a search on shodan and found more than 200 iTop instances that expose sensitive info like this.

Most notably, while investigating this we discovered that the official support instance of iTop at support.itop-saas.com was also affected under a certain condition, potentially allowing third parties to dump the whole database.

Combodo acknowledged this issue and fixed their website however they made no comment on changes to the iTop platform to prevent this in the future.

While strictly speaking this is not a vulnerability of iTop, given the widespread nature of this issue we believe that the web portal should enforce this configuration as well as periodically check this condition in order to properly notifying system administrators.

If you manage an iTop instance we urge to ensure that its web access policies are properly configured.

Disclosure timeline

- May 28th 2024: We sent our initial report to the iTop security contact
- June 4th 2024: Combodo acknowledged the issues and began working on the fixes
- November 7th 2024: The Github advisories were made public.
- February 20th 2025: We informed Combodo that we are planning to disclose our findings, they acknowledged.
- March 20th 2025: Publication of this report

DEFENCE TECH
TINEXTA GROUP

2

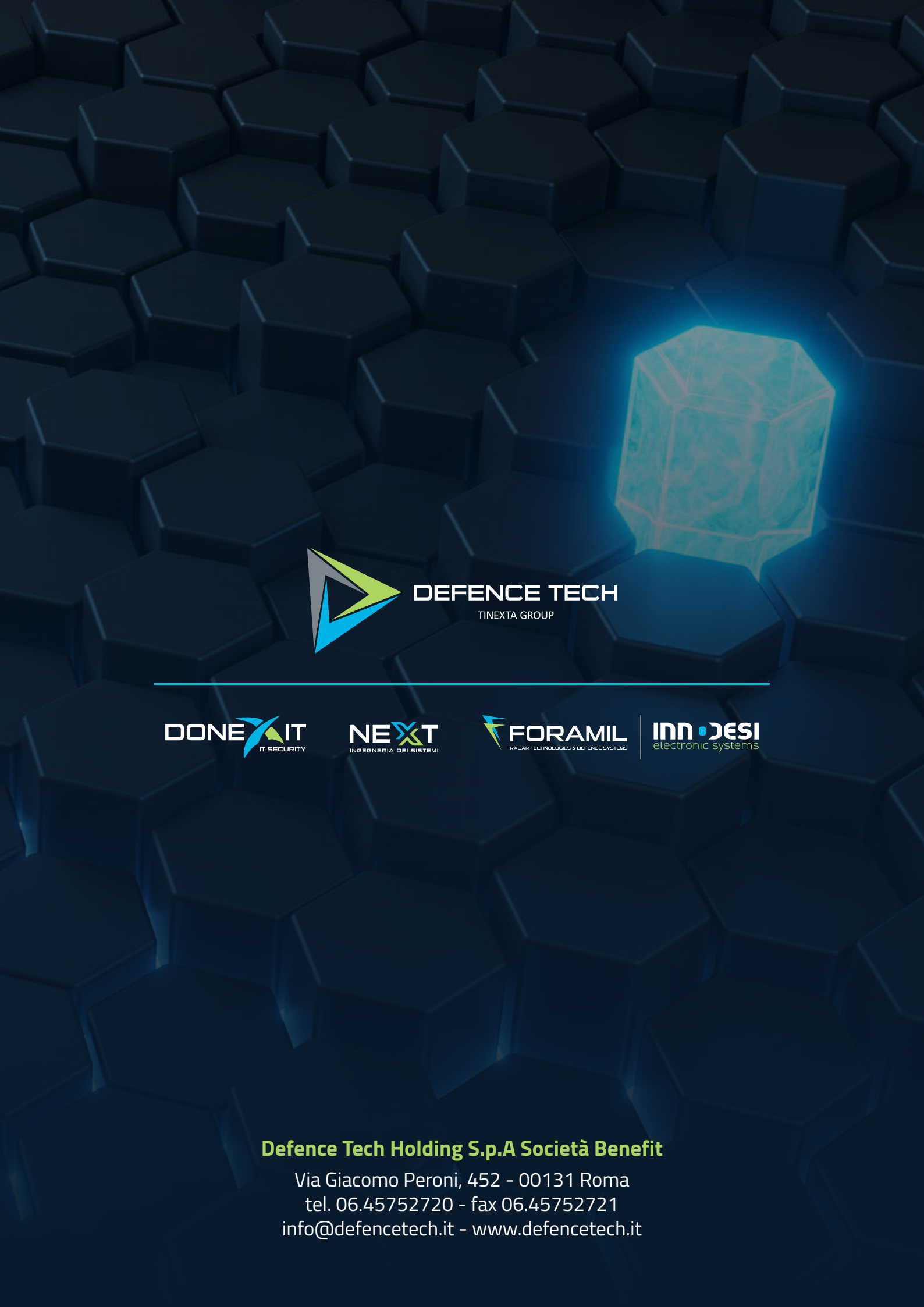
Conclusions

4. Conclusions

Security is not just a matter of using reputable software but also actively exploring the security implications of software that is in use or in the process of being adopted.

Throughout this article we explored one such case: the vulnerabilities we discovered in Combodo iTop while performing a security audit.

We recommend that iTop users update to the latest version as soon as possible and apply the suggested mitigations for the issues that have not been addressed by the vendor.



Defence Tech Holding S.p.A Società Benefit

Via Giacomo Peroni, 452 - 00131 Roma
tel. 06.45752720 - fax 06.45752721
info@defencetech.it - www.defencetech.it