

WHITEPAPER



# EXPOSING THE WEAKNESS:

The Benefits of Penetration Testing



# Table of Contents

01\_ Summary

02\_ Introduction

03\_ What is Penetration Testing and why your organization needs it

04\_ Zitec's Penetration Testing methodology & approach

05\_ Penetration Testing Case Studies

Case Study Client A

Case Study Client B

06\_ Conclusions

## SUMMARY

Over the past few years, cybercrime has become a rattling trend affecting a variety of businesses from different industries. Bad actors have exploited any surface or space where data is stored, thus constantly trying to outpace the ability to secure potential attacks. It is no wonder that Cyber Crime Statistics 2023 predicts that the global annual cost of cybercrime will reach 8 trillion USD in 2023.

Having an up-to-date security strategy and prioritizing the need to assess any risks have never been more important. Today, to keep pace with and mitigate the cybercrime epidemic, you need more than cybersecurity alerts. This is why our Security and Data Protection experts have put together a white paper tackling a strong and responsive security strategy: exposing your business' weaknesses through **Penetration Testing**. You will learn the most essential aspects and benefits of this effective solution:

What a pentest is and why it is crucial to an organization's security

What major vulnerabilities are uncovered when using this security method

What is Zitec's Penetration Testing approach

Some concrete examples of potential vulnerabilities discovered during tests conducted for our clients



Enjoy your reading!

## INTRODUCTION

**Cybercrime is a constant concern for businesses of all sizes.**

A series of recent statistics demonstrate that cyber criminals never sleep. Instead, they take advantage of any opportunity to target weaknesses in online systems, networks, and infrastructure.

### Global cybersecurity facts and figures:

In 2022, data breaches cost businesses an average of **4.35 million USD**, compared to 4.24 million USD in 2021.

The country with the largest increase in **data breaches between Q2 and Q3 2022** was China. It totaled 14,157,775 breached accounts.

**33 billion accounts** will be breached in 2023.

Cyber criminals earn **1.5 USD trillion every year** due to cybercrime.

By 2025 cybercrime will reach **10.5 trillion**.

## Think Offensively to Secure Defenses

Unauthorized access to company resources is a serious and escalating concern as attack surfaces grow through evolving business practices and IT ecosystems. Hardening new and existing applications, networks, and systems is crucial to addressing vulnerabilities before they can be exploited. Just as real-world adversaries are evolving every day to exploit new gaps, organizations must adapt testing approaches to understanding how their digital footprint would fare against skilled adversaries.

It is high time business owners embrace a proactive mindset and think outside of the box regarding any unauthorized access to their company's resources. As there are more and more IT ecosystems, security practices should also be rethought in a more evolving way to enable security experts to detect vulnerabilities before hackers do it.

Testing and exploiting your own company is a great approach to understanding the gaps in your applications, networks, and systems. This way you will be aware of how your networks might perform in the face of skilled opponents.

One such practical solution is **Penetration Testing** - a mature approach to discovering and mitigating security flaws before they actually happen. In the following lines, you will find everything you need to know about this method. Read further why penetration testing is important for your business and how Zitec's very own security specialists dealt with issues that could have escalated if exploited for real.

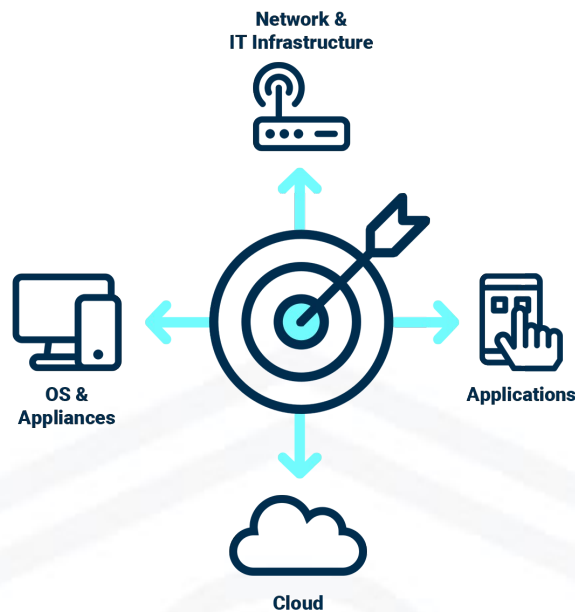
We hope this white paper will prove insightful in protecting your digital assets and adopting a proactive approach to your security practices.



# What is Penetration Testing and why your organization needs it

Penetration Testing is the process of manually testing a system component's security features and applying simulated attack scenarios to an application or infrastructure in an effort to find weaknesses and exploit them. Tests are performed from an attacker's perspective, while vulnerabilities are exploited to see how far a bad actor can get.

- ✓ Ensure the **effectiveness of your current controls**, as well as their configuration and implementation.
- ✓ Develop control mechanisms that **address weaknesses** discovered at the infrastructure, applications, processes or people level.
- ✓ **Improve incident response time** and procedures by analyzing how your teams respond to intrusions.
- ✓ Check whether the introduction of new applications or services has resulted in the **development of new vulnerabilities** that could compromise your security.
- ✓ Confirm that all your necessary security updates have been installed, new software has been safely integrated, systems are configured correctly, **operating systems aren't vulnerable to attacks, and your employees are properly trained** regarding security procedures.



While certain systems can be handled in particular ways when performing Penetration Testing, others can be addressed more generally. The most commonly addressed targets are:

### **Network & IT Infrastructures**

As external, internet-visible IT systems are regularly targeted, you should test them frequently or whenever major changes are implemented. In this case, a pentest gives you a complete overview of your exposure and shows you how to mend any compromised systems that would otherwise allow an attacker to perform lateral movement inside your infrastructure.

### **Web & mobile applications**

Web and mobile applications are often exposed to the world and not always protected very properly. Therefore, testing vulnerabilities at the device and API level, as well as the architecture, design, infrastructure, and configuration of your different types of applications will strengthen your data protection and data storage controls.

## OS & Appliances

An essential component of the security posture of all your IT environments is the setup of your operating systems (OS). Using baselines and safe builds, you can manage risks in complex settings by securely deploying servers, endpoints, and appliances.

Moreover, analyzing trust relationships, auditing procedures, and access rights and authorizations are all important elements of the security assessment.

## Cloud & hybrid infrastructures

Due to the shared responsibility of both the customer and the cloud service provider, there are new risks that need to be assessed, especially when it comes to the configuration of the cloud service. A detailed assessment of the Cloud Service Provider configuration (Azure, AWS, or Google) will help you ensure secure cloud deployments.





# Zitec's Penetration Testing methodology and approach

We leverage our methodologies and security expertise with targeted threat research to proactively identify specific areas of exposure.



## METHODOLOGY

### Pentest or Ethical Hacking

We perform in-depth manual tests and analysis on your business logic and posture. This method enables our experts to become white-hat hackers, by simulating cyber attacks from a real hacker's perspective.

### In-Depth Analysis

We identify the known gaps by using automatic scanning tools. We bring into play manual attacks, customized for the specifics of the target system, correlated with vulnerability lists such as TOP 10 OWASP or TOP 25 SANS.

### Personalized Testing

Our security experts are familiar with and perform 3 types of Penetration Testing: Black Box, Grey Box, and White Box. The testing methodology depends on the complexity of your infrastructure.

## APPROACH

01

### Information Gathering

Having critical information at the right time is the first step in a hacker's activity – both for a black- or white-hat researcher. We can be one step in front of cyber attackers by gathering information that could constitute their target.

### Reconnaissance

The more sensitive information about external network assets belonging to your company, the more the probability to obtain essential results. We check everything from servers, websites, APIs, routers, printers, employee stations, and VPNs.

02

03

### Discovery & Scanning

Via our vulnerability examination, we detect and exploit issues and weaknesses in your system. Our team checks if there are any open ports or unauthorized hosts that might target your network and business.

### Vulnerability Assessment

Offering vulnerability assessment services allow us to quantify, classify and analyze security gaps in your infrastructure and applications. In this phase, we can cover large suites of known risks and weak points.

04

05

### Exploitation

Using the data identified in the previous steps, we can compromise your website, portals, VPNs, or email services and gain unauthorized access into a network's internal system. Let us exploit these before real attackers get control over them.

### Analysis & Reporting

We analyze the data, review our findings, create a complete assessment, and report any vulnerabilities for you to reevaluate better your current infrastructure, business risks, and remediation costs.

06

# Penetration Testing Case Studies

Zitec's Security and Data Protection Department (SDP) oversees and addresses security issues related to our clients' business strategies and objectives in order to strengthen their security posture. Creating the right defense for each client requires us to be constantly up to date with the latest trends and technologies that are shaping the fast-changing security landscape.

Our security engineers are on an ongoing quest to develop, test, and implement solutions that provide product teams with the necessary input data regarding security risks. Apart from the SDP's regular engagements, the department receives routine requests from new clients.

## Case Study Client A

One of our clients, let's call him Client A, reached out to us with a bold, yet specific request. Client A wanted Zitec's Security Team to test the security of their web infrastructure in a controlled penetration exercise. The company has many applications open to the web. It was essential for them to know of any potential vulnerabilities in the network or the system.

Given the complexity of its infrastructure, **Client A found it challenging to discover the systems that were most vulnerable to outside threats** as they only had a short amount of time at their disposal.

Our team was given a high-level overview of the business-critical systems and how they are interconnected.

However, we considered that additional visibility of the existing public infrastructure was required in order to properly estimate the effort that would go into the assessment.

We decided to allocate some time for **mapping out existing framework resources** such as domains, subdomains, and IP addresses belonging to the client. Based on the results found in this phase, we scheduled a scoping meeting in which we specified what applications, systems, and IP ranges would be included in the test.

Additionally, we helped Client A **prioritize the investigation of their systems** based on a set of factors: visibility, sensitive data stored and processed, and the number of users. At the end of the scoping phase of the test, there were 3 well-defined targets to simulate an attack on.



# Targets in our crosshairs

## Administrator portal

The customers described one of the web applications that we had to assess as an administrative area where only authorized users should be able to login and use existing features. This system had the role of a **control tower** that managed the information stored and displayed on the client's main portal, also one of the targets. The administrator dashboard was hosted on a subdomain with a suggestive name: <https://admin.redacted.com/>

We opened that URL in a web browser and, at first glance, based on the message shown it seemed that access to anonymous users was restricted.

### Server Error

**403 - Forbidden: Access is denied.**

**You do not have permission to view this directory or page using the credentials that you supplied.**

For pen testers or developers, this error message may look very familiar. Based on previous experience (or by typing in the error message in Google) it was clear that the target was an ASP.NET web application. With this step, our journey in the process of testing the application began.

## Information gathering

Before starting Penetration Testing of any public-facing web application it is necessary to research the Internet presence of the target and collect as much information as possible. This initial step is called **Information Gathering** and helps in **determining various entry points that can be used for later attacks**. Usually, multiple methods are applicable in this process, but among the most popular are **port scanning, service fingerprinting, assets discovery, and subdomain brute force**.



However, being a relatively isolated target, it didn't help us much at that point and the only information we had was that it is a web application, protected by a Cloudflare WAF and without many references in search engines results.

At that point, we understood that the only way to get a foothold into the system should be through the web application itself and its exposed functionalities. We decided to not let ourselves be discouraged by the authorization error message and considered looking for other ways to find at least the administrator login page.

### Fuzzing existing files and directories

The next step was to perform a **content discovery phase**, where through brute-force techniques we **detected unreferenced, obsolete, or forgotten files and directories available on the web server**.

Payload	Status ^	Error	Timeout	Length
tool	200			7881

There are plenty of ways to achieve this, but we knew that the key in this technique is to use a fast tool (e.g. Burp Intruder, ffuf) and generous word lists.

Running the scan didn't reveal many accessible entry points, as most of the results were forbidden directories, except the following path: <https://admin.redacted.com/tool>.

Once accessed in the web browser, after a few seconds of loading, the page redirected us to another path: <https://admin.redacted.com/adminTool/tool> where we were greeted with the administrator log-in screen we were waiting for. However, the browser redirection was implemented on the client side through JavaScript, and we decided to analyze more in-depth how this functionality was implemented on the original page.

## Inspecting the source code

Inspecting the source code revealed references to external JavaScript libraries and CSS resources used for the application UI. However, through the last lines of code, we noticed that the AngularJS services and controllers responsible for the Administrator application were referenced before a successful log-in.

```
<script src="/app/app.js"></script>
<script src="/app/constants.js"></script>
<script src="/app/services/testService.js"></script>
<script src="/app/services/demoService.js"></script>
<script src="/app/controllers/administratorCtrl.js"></script>
<script src="/app/controllers/userCtrl.js"></script>
<script src="/app/controllers/loginCtrl.js"></script>
```

Exposing the client-side part of a sensitive web application before the user proves his identity could be dangerous as technical details (hostnames, API endpoints, requests descriptions) are usually stored in JavaScript files. External attackers could access useful information about existing paths, allowed methods, and request parameters and obtain a high-level overview of how the application's functionalities are built.

## JavaScript reverse engineering

A good review of previously discovered files could be useful in our efforts. Using a combination of both manual and automated inspection we took every JS file for a closer look. In just a couple of minutes, we discovered the API server's location::

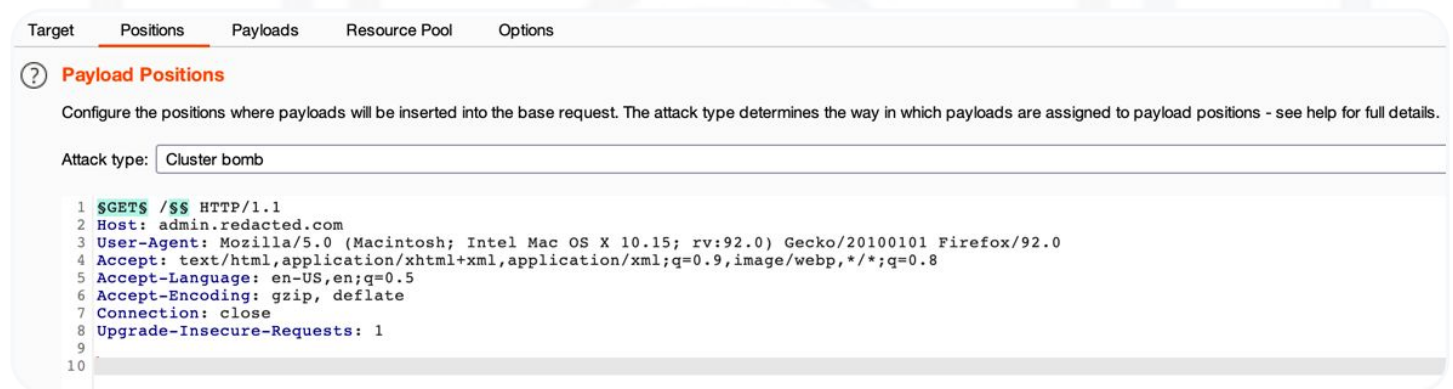
<https://admin.redacted.com/api/> as well as a whole list of API endpoints.

There were approximately 90 different endpoints and checking each of them manually was not time efficient. We then thought of a solution that would allow us to make calls to each API endpoint efficiently with the option to easily modify requests and see server responses.

We relied on the **Intruder functionality** found in Burp as we were able to use the list of endpoints and perform API fuzzing.

Thinking about how API servers are developed and how they can provide different responses depending on the method used in the request, we improved our attack by supplying another list containing all the existing HTTP methods.

The **Cluster bomb** attack type proved to be a golden bullet as we could issue requests with all existing permutations between endpoints and methods. For PUT and POST requests we needed a few additional headers, but overall, this fuzzing phase took about one minute to be completed.



Most of the requests resulted in **401 Unauthorized** responses and couldn't be used for further attacks. However, **one** of them returned a different error message, a **400 Bad Request** that for us looked promising: *api/uploadLogo*.

As the readers may guess, based on its name, **the endpoint was used for file-uploading functionality**. We wanted to see how the front-end components of the application interacted with this endpoint, so we downloaded all the AngularJS JavaScript files locally and did a quick search. We specifically looked for the presence of this endpoint, as it could be useful for us to understand which parameters, values, and headers are accepted by the application for calling this endpoint.

## Unauthorized upload to the web server

[illegible]

## EXPOSING THE WEAKNESS: The Benefits of Penetration Testing

## Out of the box

Until this part of the assessment, we knew that our target was a web application, hosted on a Microsoft-IIS server (disclosed in the Server HTTP header) and developed in ASP.NET framework. What we needed for going further was the location where all uploaded files were stored.

We decided to review all the information we had until that moment, including the files and folders discovered in the fuzzing phase. This way we remembered a directory called **Content** that denied our access every time we tried to access it. Our research found that this folder is auto-generated and specific for ASP.NET MVC applications and contains static files such as images, stylesheets, and icons. Our uploaded image was indeed a static resource and by trying to access it in that path:

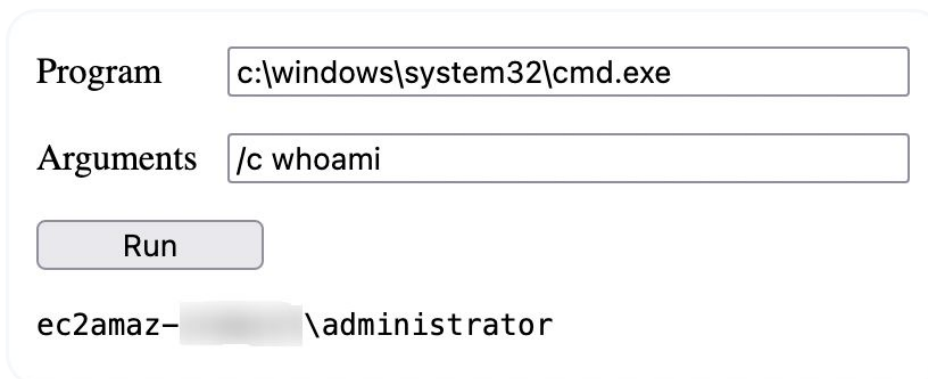
<https://admin.redacted.com/Content/img.png>, we received our photo back.

## Uploading a shell

Once an attacker finds a vulnerability, he usually doesn't stop until he elevates his privileges and infiltrates more into the target system. That's what we did in our further steps: we decided to upload files that may allow us to execute server-side code.

Based on the knowledge we had until that moment of the application, we specifically looked for those files that would allow us to run Visual Basic code, as they would have had the most chances to be interpreted correctly by the web server. We gathered a list of 16 file extensions, powered on the Intruder, and started the upload process. In the end, one file with an .aspx extension was found to be allowed by the web application. We uploaded a simple shell interface containing two components: one that executed the system commands in the backend and one which returned the results in the web browser. Running a **whoami command** disclosed that we had the Administrator role over the machine we'd just compromised.





The screenshot shows a Windows command prompt window. The 'Program' field contains 'c:\windows\system32\cmd.exe' and the 'Arguments' field contains '/c whoami'. A 'Run' button is visible below the input fields. The command prompt output shows 'ec2amaz- [redacted] \administrator', indicating the user is running as an administrator on an Amazon EC2 instance.

Based on the result of the **whois command**, the machine was an Amazon EC2 instance that possibly could give us more permissions in the AWS infrastructure. However, when we tried to query the metadata service, we received some internal errors that would require us to modify the system configuration. At that point, it was clear to us that we had reached our goal.

As soon as the Penetration Testing exercise was finished, **our security team delivered a report to the client containing all the security vulnerabilities**. This has allowed them to gain an attacker's perspective on their biggest vulnerabilities.

The Security and Data Protection team provided expert consultancy and digital perimeter protection, including:

- Expert consultancy for threat prevention
- Protected digital environment
- Safe and user-friendly experience on the app
- Improved security controls
- Quick responses to cyber attacks

Looking at the bigger picture of this case, our team outlined the following security best practices:

✓ **Web applications** specifically designed for internal use but need to be publicly available **should not disclose resource files** (scripts, stylesheets, static content) before the user logs-in.

✓ Authorization and access-control mechanisms should be implemented for each API endpoint; **users should have dedicated roles, and access to the application's functionalities should be granted based on granular security policies.**

✓ **Upload functionalities should be designed to consider the possible security misconfigurations** that may appear (e.g. user uploads files with different extensions and content types).

✓ **User-uploaded content should be stored in sandboxed locations** where execution rights are not possible (e.g. storage buckets).

✓ **The web service should run under a limited system account** that has access only to files and folders used in the context of the web application.

# Case Study Client B

**Another client, let's call him Client B,** acknowledged the potential threats and contacted us for a full security audit composed of a White-Box Penetration Testing assessment for its main applications: customers and administrative platforms.

Our client's infrastructure, composed of several web applications and web APIs, exposes functionalities for product presentation, user management, order flows, payment services, and administrative portals.

Due to the sensitivity of the data that travels through the systems included in its ecosystem, the client was prone to potential cyber-attacks.

## White-Box Penetration Testing

Typically, a white-box pentest for a web application is an assessment that includes two flavors of testing: attacking the application in a controlled environment and performing

code review directly on the application's source code. This approach is a little different from the classical way of performing Penetration Testing as it **includes unrestricted access to a critical part of the application that an attacker usually doesn't have access to**. From the security assessor's perspective, having the possibility to read the code line by line and understand the application flows represents an easier way to spot security issues but increases the complexity and necessary time to complete the audit.

As resources allocated for this kind of assessment are usually located in clearly defined boundaries, during our previous code review projects we defined a framework that proved to be efficient in terms of time spent and identified vulnerabilities.

As a general overview, we apply **standardized methodologies such as OWASP Code Review Guide or OWASP Web Application Testing Guide** and maintain a set of specific vulnerabilities depending on the technology. However, each application is different in terms of architecture and components, and following a set of recognized security best practices such as **Application Security Verification Standard** was helpful for us in distributing the allocated time. To this end, the effort is split equally and the assessment is performed in an organized fashion, with each architectural component such as Authentication, Authorization, Input Validation, or Session Management being reviewed.

Making a step back to our customer's case, the application code was hosted in a Gitlab cloud instance. First, we wanted to obtain a general overview of the code and application security before actually digging into the code, so we attached our custom [360 Security Solution](#) to the CI/CD deployment, as it integrates smoothly with external clients.

In this project, the following scanning engines were enabled:

Static Application Security Scanning (SAST)

Software Composition Analysis

Secrets Detection

After collecting and reviewing the results, we moved further and started to manually review the code. The first step was to observe the structure of the repository and the build configuration files, which revealed the technologies used: NodeJS, ExpressJS, MySQL, and others. Then we took a look at how the application scripts were organized and noticed the MVC framework that was followed.

Being in the face of a web application, we decided to make a fast check over the application routes as they serve as the main entry point. We observed how a request travels the following path: route → controller → repository → view and how different components were tied together.

### The path to the database

One of the most crucial aspects of a web application security strategy is the way it handles user input, as gaps in this control could open serious doors for attackers nearby. Input Validation is one of the components we never skip when we're in front of a source code.

**SQL Injection** is a vulnerability that some of us may say it's a long-gone issue as more and more web frameworks and libraries come in to help developers with security features such as parameter binding, query builders, or object-relational mappings. The same case applies to our client, as it was using one query builder called **Knex.js**.

**Knex.js** is a query builder for SQL databases; it allows for building and executing SQL queries in a convenient, programmatic way, with a simple and clean API. Regarding protection against SQL injection, it uses parameterized queries to safely escape user input, which effectively prevents malicious users from injecting arbitrary SQL code into a query. This makes it more secure compared to building SQL strings manually, which is prone to SQL injection attacks.

During our analysis, we noticed many of the SQL queries contained user input that were not filtered or escaped in any way:



```
const db = require('@db');
const orderId = req.query.id;

try {
  const orderInformation = await getOrderInformation(orderId);
} catch (OrderInformationError){
  throw new Error(OrderInformationError);
}

const getOrderInformation = (orderId) => db
  .withSchema('orders_schema')
  .select('orders.*')
  .from('orders')
  .innerJoin('users', 'orders.user_id', 'users.id')
  .where({ 'orders.id' : orderId})
  .first();

module.exports = { getOrderInformation };
```

Unfortunately for us, placing a malicious payload in such constructions of the builder did not disclose any security vulnerability, as input data was automatically escaped. We took almost every query for an in-depth analysis but there were absolutely no chances to alter any of the SQL statements.

We thought that we should move on and consider that developers carefully integrated the Knex.js framework into their application. However, right when we were closing all the repository files, an interesting idea came to our minds: **what if there are any ways developers can bypass the security mechanisms included in Knex.js framework and write custom queries with input data coming from the user?** We decided to give it a try and started to read the documentation: <https://knexjs.org/guide/query-builder.html>

In this way we found out about a feature of the framework, called Raw Queries. These constructions can be handy, if you are a developer and you need to write some complicated queries that can't fit in the regular builder. However, from a security perspective, using them can expose the application to SQL injection attacks, as the defense mechanisms are disabled and developers should perform the parameter binding manually. We thought that this might be our chance to find a serious issue, so we constructed a list of all the existing raw query statements:

knex.raw()  
knex.fromRaw()  
knex.whereRaw()  
knex.joinRaw()  
knex.orderByRaw()  
knex.havingRaw()

We then started to look exclusively for these statements and, after a while, we discovered the following code:

```
const db = require('@db');
const inputCode = req.query.code;

try {
  const promoCode = await getCode(inputCode);
} catch (err) {
  return res.status(500).json({
    success: false,
    type: 'error',
    message: 'Internal server error.',
    data: null
  });
}

const getCode = async (code) => {
  const result = await db
    .from('PromotionalCodes')
    .join('Categories', 'PromotionalCodes.category_id', 'Categories.id')
    .whereRaw('UPPER(PromotionalCodes) = `${code.toUpperCase()}`')
    .where('PromotionalCodes.expired', false)
    .first(
      'PromotionalCodes.code',
      'PromotionalCodes.amount',
      'PromotionalCodes.validity',
      'Categories.name'
    );
  return result;
}
```

As it can be seen, this query was trying to retrieve some information from the database. Still, the **whereRaw** clause was constructed using a parameter called **code** that contained raw data, coming directly from the user, without binding it. In a few words, the path for an exploitable **SQL Injection vulnerability**. We knew we should verify this finding so we followed the data flow until we reached the application endpoint that was exposing the corresponding functionality:

**<https://application.target.com/check-code?code=<code>>**. It was a public route, integrated into the main web application.

At this point, we knew this might be a critical finding so we quickly put the [SQLMap](#) tool to work by carefully configuring all the necessary arguments:

```
sqlmap.py -u https://application.target.com/check-code?code=1 -p "code"
```

In just a few moments, our assumption was confirmed: we found **a critical vulnerability in the application customer portal**. As always, during a Penetration Testing assessment, it's important to correctly assess the risk of the findings and estimate the worst-case scenario. In our case, we started to look for interesting tables in the database and found the **users** and **admins** entries.

We knew that the user's table contains all the customer information; however, an attacker is always interested in getting the highest access level. As such, being in a controlled (staging) environment, we listed the first few lines from the **admins** table that revealed the majority of the admin users along with sensitive fields such as **id**, **name**, **email**, and **password**, but the least being in a hashed form. However, we noticed an additional field in the result called **reset\_code**. We immediately thought that this code might be the reset code generated after the Forgot Password functionality is used by an admin who can't access his account. After testing this assumption, we discovered that we were right, so we issued a new reset code for an existing administrator and managed to log in.

We continued to look for this kind of SQL injection in the remaining code, but we couldn't find any. Carefully investigating issues in this component, we decided to move to the remaining components of the architecture so that we can have a bigger picture and possibly more vulnerabilities.

At the end of the assessment, we aggregated the findings identified during code review, the ones obtained while attacking the application, and those reported by our scanning engines into a comprehensive report presented to the client. Communication with our client was essential in the assessment process , which helped us quickly solve critical issues.

The White-Box Penetration Testing we conducted for our client revealed serious security flaws, despite the carefully developed code, written using secure frameworks. Moreover, our team has delivered cyber security training for our client's development team to improve developers' general security knowledge across different technologies.

Following this **White-Box Penetration Testing assessment**, our team has identified some important security recommendations that companies can apply to ensure that applications stay protected from potential threats:

- ✓ Development teams should **ensure that all security features and configurations are fully understood and implemented correctly** to avoid potential vulnerabilities
- ✓ **Regular security assessments**, including white-box penetration testing, **should be conducted to identify any security issues** that may have been missed during the development process
- ✓ Developers should **follow secure coding practices**, such as input validation and output encoding, **to prevent common vulnerabilities** such as Cross-Site-Scripting (XSS) and SQL injection
- ✓ **Security testing should be integrated** into the Software Development Life Cycle (SDLC) to ensure that security is considered at every stage of development

# Conclusions

While digital transformation means huge value in terms of savings and efficiency for your business, it also opens the list of vulnerabilities to your digital assets. A system is only as secure as its weakest link and a penetration test is the first step in assessing the security level of your system or network.

Given the widespread impact of the recent cyberattacks, companies in every industry shift from defensive to proactive approaches and learned to leverage Penetration Testing to level up their security posture. Ensuring this proactive approach can be overwhelming, especially for companies with complex infrastructures. However, a dedicated, CREST-certified security partner can efficiently address even complex systems and not only reveal vulnerabilities, but also help you strengthen your defense mechanisms.





# About Cyber Security Penetration Testing

Our security experts review, exploit, and report everything and anything so that you find peace of mind when it comes to anything security-related. Moreover, our CREST accreditation is a confirmation that the penetration tests we're running are backed by proper and up-to-date skills, strategies, and techniques to give you the best assessment of your cybersecurity posture.

If you would like to know how prepared your company is in the event of an attack, our Security and Data Protection team can shine a light on the loose ends. As a one-stop shop for security, we can develop the right security strategy to keep your assets out of harm's way.

**Drop us a line and schedule a meeting to help your organization stay safe.**

**[CONTACT US](#)**

# About Zitec

One of Europe's largest and most prominent end-to-end software development services companies, Zitec is the digital transformation partner to companies across over 30 countries, such as the USA, Canada, the UK, Romania, Cyprus, Germany, Italy, Denmark, and the Middle East.

Zitec is one of the few Romanian companies certified Google Cloud Premier Partner, Microsoft Solution Partner, Amazon AWS Technology Partner, and provider of the VTEX cloud eCommerce solution platform, as well as Adobe Solution Bronze Partner. The company is ISO:9001, ISO:27001, CREST certified, and acknowledged by DNSC as a NIS Security Auditor. Furthermore, The Manifest recognized Zitec as one of the Global Dev & IT industry frontrunners for 2022, being one of the most reviewed Cloud Consultants in the world.

All information contained herein is property of Zitec.

You agree not to use, reproduce, distribute or create derivative works based on any (portion of) Zitec copyrighted work, without first receiving Zitec's express written permission.

(C) 2023 Zitec.

