BUGFYND · APPLICATION SECURITY

Thick Client Application Penetration Testing Report

Prepared for: Sample Client Pvt Ltd
Application: BetaFast (three-tier .NET client with web service)

Version 1.1 · Final report

SAMPLE REPORT: findings come from a deliberately vulnerable lab application, not from a real client

Confidentiality statement

This document is confidential and intended only for the client named on the cover. It may not be copied, shared or published, in whole or in part, without written permission from the client and Bugfynd. The client may share it with its own auditors, customers and partners to support its security and compliance programs.

About this sample: this report shows the format and level of detail Bugfynd delivers after a thick client penetration test. All findings come from BetaFast, an open-source, intentionally vulnerable three-tier .NET application (NetSPI thick client lab) tested in our lab. Client, dates and evidence are illustrative; secrets are redacted.

Contents

#	Section
1	Document details
2	Introduction, scope and duration
3	Executive summary and risk ratings
4	Approach and methodology
5	Results overview and high-level recommendations
6	Detailed findings (6)
7	Conclusion
A–C	Annexures: changes to environment, tools used, tests performed

SAMPLE

1. Document details

Document control

Document title	Thick Client Penetration Testing Report
Classification	Confidential – Final report
Report version	1.1 (includes retest results)

Document history

Version	Date	Prepared by	Status
1.0	[Date]	Bugfynd testing team	Initial report
1.1	[Date]	Bugfynd testing team	Final report after retest

Client information

Company	Sample Client Pvt Ltd
Point of contact	[Name, title]
Email	[contact@client.example]

Bugfynd team

Name	Role	Responsibility
Consultant A	Lead Consultant	Testing and report preparation
Consultant B	Security Consultant	Technical review
Consultant C	Security Consultant	Testing support and retest
[Client contact]	Client representative	Report acceptance

SAMPLE

2. Introduction

Bugfynd performed a manual penetration test of the client's three-tier Windows application, where the desktop client talks to a backend web service over HTTP to find security weaknesses an attacker could exploit, measure their business impact and give clear steps to fix them. Testing was carried out under written authorization from the client and within the agreed rules of engagement.

Scope

Application	Type	Version	Components
BetaFast	.NET WPF Windows desktop (three-tier)	[x.y]	BetaFast.exe, configuration files, backend web service (API), local cache

Duration

Activity	Dates
Initial assessment including reporting	[Start date] – [End date]
Retest and report update	[Retest date]

Test approach

- Grey-box testing with standard user and administrator test accounts provided by the client.
- Testing performed from Bugfynd's lab environment on a dedicated Windows test machine, with client traffic routed through Burp Suite to inspect and modify requests to the web service.
- No denial-of-service or destructive testing was performed.

Limitations

- Results reflect the application version and environment available during the test window.
- Server-side components were tested only where reachable from the client application.

SAMPLE

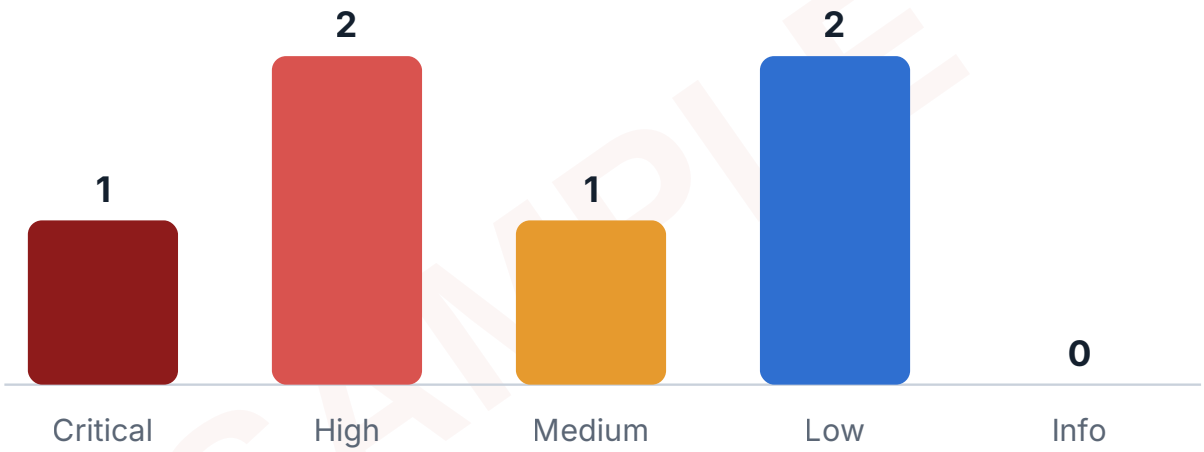
3. Executive summary

Bugfynd identified **6 security findings** in the application: **1 Critical, 2 High, 1 Medium and 2 Low**. The most serious issue lets a customer change the price of an order to any value, because the web service trusts the price sent by the desktop client. Traffic between the client and the server is also unencrypted, exposing login credentials and session tokens on the network.

After the client's fixes, a retest confirmed that **5 of 6** findings are remediated. One High finding (cleartext HTTP traffic) needs server certificate deployment and remains open with an agreed plan.

1	2	1	2	0
Critical	High	Medium	Low	Info

Findings by severity



Initial test vs retest

Status	Critical	High	Medium	Low	Total
Initial assessment	1	2	1	2	6
Open after retest	0	1	0	0	1

SAMPLE

Risk rating

Each finding is scored with **CVSS v3.1** (Common Vulnerability Scoring System), the industry standard for rating how severe a vulnerability is. Scores run from 0 to 10.

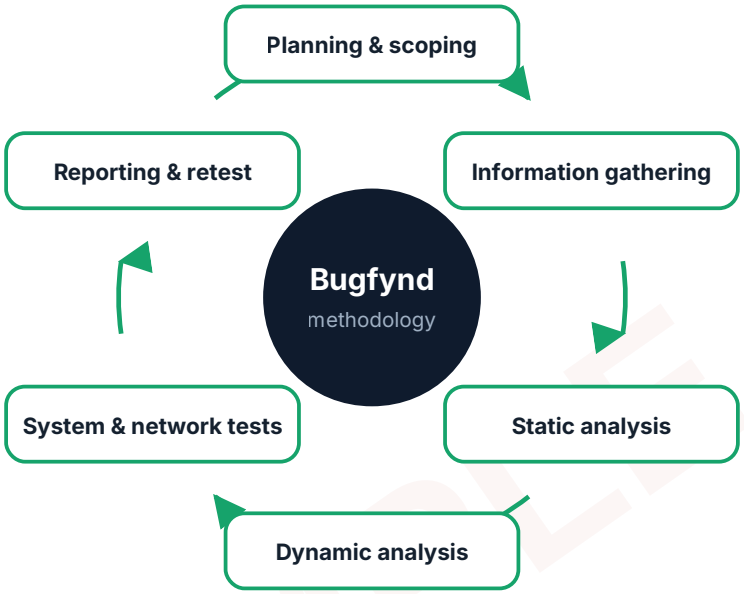
Severity	CVSS score	What it usually means
Critical	9.0 – 10.0	Easy to exploit, often remotely and without credentials; can fully compromise the application or its data. Fix immediately.
High	7.0 – 8.9	Exploitable with little effort; gives access to sensitive data or higher privileges. Fix as a priority.
Medium	4.0 – 6.9	Needs specific conditions or some access; limited impact on its own but can be chained. Fix in the next release.
Low	0.1 – 3.9	Minor impact or hard to exploit; mainly defence-in-depth. Fix as part of normal hardening.
Info	0.0	Best-practice observation with no direct security impact.

Scores are calculated with the FIRST CVSS 3.1 calculator (first.org/cvss/calculator/3.1) and adjusted for the client's environment where relevant.

SAMPLE

4. Approach and methodology

Bugfynd's thick client methodology combines manual testing with tool support and follows recognised standards: **OWASP Desktop App Security Top 10**, **OWASP Top 10**, **OWASP API Security Top 10**, **CWE Top 25** and the **Penetration Testing Execution Standard (PTES)**.



Phase	What we do
Planning & scoping	Agree scope, test accounts, testing windows and contacts; obtain written authorization.
Information gathering	Map the architecture, technologies, entry points, files, registry use, network endpoints and dependencies.
Static analysis	Decompile binaries (dnSpy, ILSpy), review configuration files and stored data, look for hard-coded secrets and weak cryptography, check binary protections.
Dynamic analysis	Test inputs for injection, authentication and authorization flaws, GUI control bypass, file handling and business logic while the application runs.
System & network tests	Inspect memory, file and registry permissions, DLL loading and traffic (TLS, cleartext protocols, request tampering).
Reporting & retest	Document every finding with impact, steps, evidence and fixes; report critical issues the same day; retest after fixes.

Communication

- Daily status updates during testing.
- Critical findings reported to the client contact the same day.
- Walkthrough call with the development team after report delivery.

SAMPLE

5. Results overview

#	Finding	Severity	CVSS	Retest status
1	Price Manipulation in Order API	Critical	9.1	Remediated
2	Cleartext HTTP Between Client and Web Service	High	7.4	Open
3	Insecure Direct Object Reference on Orders (BOLA)	High	7.1	Remediated
4	Session Token Not Invalidated on Logout	Medium	5.4	Remediated
5	Verbose Error Messages Reveal Internal Details	Low	3.7	Remediated
6	Sensitive Data in Local Cache Files	Low	3.3	Remediated

High-level recommendations

1. Treat the desktop client as untrusted: recalculate prices and check permissions on the server for every request.
2. Serve the web service only over HTTPS with TLS 1.2 or higher, and validate the certificate in the client.
3. Enforce object-level authorization on every API endpoint.
4. Return generic error messages and log details on the server only.
5. Expire sessions on logout and after inactivity; use random, unpredictable tokens.

SAMPLE

6.1 Price Manipulation in Order API

Critical

Severity / CVSS	Critical · 9.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H`
Affected component	POST /api/orders (web service)
Category	CWE-602
Retest status	Remediated
Description	When a customer places an order, the desktop client sends the item price in the request. The web service saves the order with that price without checking it against the product catalogue.
Impact	Any customer can buy items at any price, including zero, causing direct financial loss.
Steps to reproduce	1. Route BetaFast traffic through Burp Suite. 2. Place an order and intercept the POST /api/orders request. 3. Change the price field to 0.01 and forward the request: the order is accepted at that price.

Evidence	`POST /api/orders`
	`{"itemId": 42, "quantity": 1, "price": 0.01}`
	`HTTP/1.1 200 OK  {"status":"confirmed","total":0.01}`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Ignore client-supplied prices; look up the price on the server. • Validate quantity, discount and totals on the server. • Alert on orders whose totals differ from catalogue prices.
Reference	https://cwe.mitre.org/data/definitions/602.html

SAMPLE

6.2 Cleartext HTTP Between Client and Web Service

High

Severity / CVSS	High · 7.4
CVSS vector	`CVSS:3.1/AV:A/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:N`
Affected component	Client ↔ web service communication
Category	CWE-319
Retest status	Open
Description	All communication between the desktop client and the web service uses plain HTTP, including login requests and session tokens.
Impact	An attacker on the same network can capture credentials and session tokens, take over accounts and change requests in transit.
Steps to reproduce	1. Start Wireshark on the test machine. 2. Log in to BetaFast. 3. Filter on `http` and open the login request: the username and password are readable.

Evidence	`POST http://[api-host]/api/login`
	`{"username":"testuser","password":"[REDACTED – visible in cleartext]"}`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Serve the web service only over HTTPS with TLS 1.2+. • Validate the server certificate in the client (consider certificate pinning). • Redirect or block plain HTTP on the server.
Reference	https://cwe.mitre.org/data/definitions/319.html

SAMPLE

6.3 Insecure Direct Object Reference on Orders (BOLA)

High

Severity / CVSS	High · 7.1
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:L/A:N`
Affected component	GET /api/orders/{id}
Category	CWE-639
Retest status	Remediated
Description	The order details endpoint returns any order whose ID is requested, without checking that it belongs to the logged-in user. Order IDs are sequential.
Impact	Any customer can read other customers' orders, names and delivery addresses by changing the order ID.
Steps to reproduce	1. Log in as test user A and open an order; note the request GET /api/orders/1005. 2. Send it to Burp Repeater and change the ID to 1004 (an order of test user B). 3. The response returns B's order details.

Evidence	`GET /api/orders/1004  (session of User A)`
	`HTTP/1.1 200 OK  {"orderId":1004,"customer":"[User B]","address":"[REDACTED]"}`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Check that the order belongs to the authenticated user on every request. • Use non-guessable identifiers (UUIDs) as defence in depth. • Add automated authorization tests for each endpoint.
Reference	https://owasp.org/API-Security/editions/2023/en/0xa1-broken-object-level-authorization/

SAMPLE

6.4 Session Token Not Invalidated on Logout

Medium

Severity / CVSS	Medium · 5.4
CVSS vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N`
Affected component	Session management (web service)
Category	CWE-613
Retest status	Remediated
Description	Clicking Logout only clears the token in the desktop client. The server keeps the token valid, and tokens do not expire for several days.
Impact	A stolen token (see finding 2) can be reused long after the user logs out.
Steps to reproduce	1. Log in and copy the session token from Burp. 2. Click Logout in the client. 3. Replay an authenticated request with the old token: it still succeeds.

Evidence	`After logout: GET /api/profile  Authorization: Bearer [old token]`
	`HTTP/1.1 200 OK`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Invalidate tokens on the server at logout. • Set short token lifetimes and an inactivity timeout. • Rotate tokens after login and privilege changes.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

SAMPLE

6.5 Verbose Error Messages Reveal Internal Details

Low

Severity / CVSS	Low · 3.7
CVSS vector	`CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N`
Affected component	Web service error handling
Category	CWE-209
Retest status	Remediated
Description	Malformed requests return full .NET stack traces, including internal file paths, framework versions and database details.
Impact	Attackers learn about the server's technology and structure, which makes other attacks easier.
Steps to reproduce	1. Send a request with an invalid JSON body to POST /api/orders. 2. Review the error response.

Evidence	`HTTP/1.1 500 Internal Server Error`
	`System.Data.SqlClient.SqlException ... at`
	`BetaFast.Api.OrdersController ... in [internal path]`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Return generic error messages to clients. • Log full error details on the server only. • Disable detailed errors in production configuration.
Reference	https://cwe.mitre.org/data/definitions/209.html

SAMPLE

6.6 Sensitive Data in Local Cache Files

Low

Severity / CVSS	Low · 3.3
CVSS vector	`CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N`
Affected component	%LOCALAPPDATA%\BetaFast\cache
Category	CWE-312
Retest status	Remediated
Description	The client caches the user's profile and recent orders as plain JSON files in the user's AppData folder, and the files remain after logout.
Impact	Another user or malware on the same machine can read personal and order data.
Steps to reproduce	1. Log in, browse orders, then log out. 2. Open %LOCALAPPDATA%\BetaFast\cache and view the JSON files.

Evidence	`profile.json:`
	`{"name":"[REDACTED]","email":"[REDACTED]","phone":"[REDACTED]"}`
	Screenshots and full request/response logs are attached in the evidence pack.

Recommendation	• Avoid caching personal data, or encrypt it with Windows DPAPI. • Delete cache files on logout.
Reference	https://cwe.mitre.org/data/definitions/312.html

SAMPLE

7. Conclusion

The web service trusted values and decisions made by the desktop client. The most urgent issues, price manipulation and access to other users' orders, have been fixed and verified. Bugfynd recommends enabling HTTPS for all client-to-server traffic and retesting before the next release.

Annexure A: Changes to environment

No permanent changes were made. Test orders created during price manipulation testing were cancelled and reported to the client.

Annexure B: Tools used

Tool	Purpose
dnSpy / ILSpy	Decompiling and debugging .NET binaries
Sysinternals (Procmon, Process Explorer, AccessChk)	Process, file, registry and permission analysis
Process Hacker	Memory inspection and process dumps
Wireshark	Network traffic capture and analysis
Echo Mirage / Burp Suite	Intercepting and modifying application traffic
PESecurity	Checking binary protections (ASLR, DEP, CFG)
Regshot	Comparing registry changes
sslscan	TLS configuration review

SAMPLE

Annexure C: Tests performed

Test area	Status
Information gathering and architecture review	Tested
Authentication and authorization	Tested
Input validation and injection	Tested
GUI control and object permissions	Tested
File and configuration storage	Tested
Registry storage and permissions	Tested
Hard-coded secrets and cryptography	Tested
Memory analysis	Tested
DLL loading and binary protections	Tested
Network communication and TLS	Tested
Traffic interception and tampering	Tested
Error handling and information disclosure	Tested
Business logic	Tested

SAMPLE