CES Service Call Backend Audit: ServiceSubmit.asp

Purpose:
This document supplements the main CES service call form audit and captures the backend logic and flow of the legacy ServiceSubmit.asp file, which is responsible for processing and handling all form submissions from servicecall.htm. It provides a technical breakdown, an audit of potential vulnerabilities, and modernization guidance to assist any future developers in rebuilding or maintaining this critical workflow.

Overview of Functionality

ServiceSubmit.asp performs the following key tasks:

1. **Receives POST data** from the service call form.
2. **Processes and sanitizes inputs** using FixQuotes().
3. **Generates two types of customer IDs** based on client type (personal or business).
4. **Determines default contact phone** and service options.
5. **Sends notification email** to internal CES staff.
6. **Sends confirmation email** to the client.
7. **Displays a confirmation page** showing submitted data.

Data Flow

- Accepts POST data from inputs such as:
    - Personal info (name, address, email, phone numbers)
    - Service info (onsite, drop-off, emergency, date/time preferences)
    - Hardware/software configuration
- Populates variables such as vFName, vOrg, vReqSrvDate, etc.
- Uses logic to determine:
    - Which phone number to use
    - Whether the client is new

    - Whether the service is an emergency or requires pickup
    - Which payment type was selected
- Constructs a detailed HTML email with all data
- Sends:
    - Internal email to: technician@cesitservice.com
    - CC: richr@cesitservice.com, lisa@cesitservice.com
    - Confirmation email to client

Security and Functional Audit

Area	Status / Risk	Notes / Recommendations
Input Validation	Weak	Add strict server-side field validation (length, type, required fields)
Sanitization	Partial (FixQuotes())	Confirm FixQuotes() handles XSS, SQL injection, and malformed characters
CAPTCHA / Anti-Spam	Missing	No reCAPTCHA, no bot protection. Add reCAPTCHA v2 or hCaptcha server-side validation
Logging / Archiving	Missing	Emails are sent but nothing is stored (no DB, no file). Add CSV logging or DB write before sending email
Email Formatting	Plain HTML only	Consider templating or centralizing email generation for maintainability
Rate Limiting	Missing	Add basic per-IP limits to reduce spam/flooding
SMTP Configuration	Unknown	Ensure authenticated SMTP is used with SPF/DKIM support
IP Logging	Missing	Capture REMOTE_ADDR and store/log it for traceability
Error Handling	Minimal	No user-visible error reporting or fallback actions

Area	Status / Risk	Notes / Recommendations
Sensitive Info Handling	Weak	Emailing full submissions is risky; ideally, only log to secure DB

Modernization Recommendations

If this file were to be rebuilt in a modern backend (e.g. Python Flask, Node.js Express, PHP, or ASP.NET), we recommend:

- Use a templating engine for email construction (e.g., Jinja2, Handlebars)
- Store all submissions in a secure database (e.g., PostgreSQL, MySQL)
- Authenticate outgoing SMTP and implement SPF/DKIM/DMARC
- Add server-side reCAPTCHA verification
- Split responsibilities into: validation → storage → email → confirmation
- Secure error and success logging
- Use HTTPS and sanitize all user input deeply

Next Steps for Developer Handoff

✅ **Documented So Far:**

- servicecall.htm (entire structure and purpose)
- checkform.html (alternate contact form with CAPTCHA)
- ServiceSubmit.asp (core backend handling)

🧩 **Still Needed:**

- **incs/functions.asp**: Contains FixQuotes(), sendmail(), and createCustID() logic. Must be reviewed to:
    - Verify data sanitization is secure
    - Confirm email function uses secure headers / SMTP
    - Ensure Customer ID generation is consistent and safe

- **Mail Server Configuration**:
    - Confirm SMTP server is secured, authenticated, and DKIM/SPF/DMARC records are present.
- **Email Deliverability Logs**:
    - Are messages landing in inbox or flagged as spam? Need feedback loop or bounce handling?
- **Access Control**:
    - Is this form public on the web? If so, spam protections are critical.

🛠 Suggested Immediate Dev Tasks:

1. Implement reCAPTCHA on both checkform.html and servicecall.htm.
2. Review and improve FixQuotes() function.
3. Store submissions locally (e.g. .csv, SQLite, or JSON flatfile) as a backup.
4. Set up SMTP authentication + SPF/DKIM if missing.
5. Build modern version of the backend in Flask, Node.js, or ASP.NET Core (optional but advised).

Final Note

This audit closes the loop on the legacy workflow and serves as a blueprint for future redevelopment. Any developer reviewing this should now have full visibility into both the frontend form logic and backend submission handler. Immediate priority is securing inputs and submissions, then iteratively migrating to a modern stack for long-term maintainability.

-- End of Document --