

● RELATÓRIO DE VAZAMENTOS DE DADOS MULTI-TENANT

Data: 2025-10-14

Branch: refactor/code-cleanup

Severidade: CRÍTICA

🎯 Sumário Executivo

Identificados **7 vazamentos críticos** de dados multi-tenant que permitem acesso não autorizado a dados de outras empresas.

● Problemas Críticos Identificados

1. dashboard.service.ts - 2 vazamentos

Localização: backend/src/dashboard/dashboard.service.ts

Problema 1.1 - getEstatisticas (linhas 100-103)

```
const produto = await this.prisma.produto.findUnique({
  where: { id: item.produtoId },
  select: { id: true, nome: true, imagem: true, preco: true },
});
```

Impacto: Busca produtos sem filtrar por empresaId. Pode retornar produtos de outras empresas.

Solução:

```
const produto = await this.prisma.produto.findFirst({
  where: { id: item.produtoId, empresaId },
  select: { id: true, nome: true, imagem: true, preco: true },
});
```

Problema 1.2 - getProdutosPopulares (linhas 220-230)

```
const produto = await this.prisma.produto.findUnique({
  where: { id: item.produtoId },
  select: { id: true, nome: true, imagem: true, preco: true, categoria: true },
});
```

Impacto: Mesmo problema - busca produtos sem filtrar por empresaId.

Solução:

```
const produto = await this.prisma.produto.findFirst({
  where: { id: item.produtoId, empresaId },
  select: { id: true, nome: true, imagem: true, preco: true, categoria: true },
});
```

2. avaliacoes.service.ts - 4 vazamentos

Localização: backend/src/avaliacoes/avaliacoes.service.ts

Problema 2.1 - create (linhas 12-14)

```
const produto = await this.prisma.produto.findUnique({
  where: { id: createAvaliacaoDto.produtoId },
});
```

Impacto: Permite criar avaliações para produtos de outras empresas.

Solução: Adicionar empresald ao controller e passar para o service:

```
// Controller
@Post()
create(@Body() createAvaliacaoDto: CreateAvaliacaoDto, @Request() req) {
  return this.avaliacoesService.create(createAvaliacaoDto, req.user.sub, req.user.empresaId);
}

// Service
async create(createAvaliacaoDto: CreateAvaliacaoDto, usuarioId: string, empresaId: string) {
  const produto = await this.prisma.produto.findFirst({
    where: { id: createAvaliacaoDto.produtoId, empresaId },
  });
  // ...
}
```

Problema 2.2 - findByProduto (linhas 43-68)

```
async findByProduto(produtoId: string) {
  const avaliacoes = await this.prisma.avaliacao.findMany({
    where: { produtoId },
    // ...
  });
}
```

Impacto: Não valida se o produto pertence à empresa do usuário. Permite ver avaliações de produtos de outras empresas.

Solução: Adicionar validação de empresald:

```
// Controller
@Get('produto/:produtoId')
findByProduto(@Param('produtoId') produtoId: string, @Request() req) {
  return this.avaliacoesService.findByProduto(produtoId, req.user.empresaId);
}

// Service
async findByProduto(produtoId: string, empresaId: string) {
  // Verificar se produto pertence à empresa
  const produto = await this.prisma.produto.findFirst({
    where: { id: produtoId, empresaId },
  });

  if (!produto) {
    throw new NotFoundException('Produto não encontrado');
  }

  const avaliacoes = await this.prisma.avaliacao.findMany({
    where: { produtoId },
    // ...
  });
  // ...
}
```

Problema 2.3 - findByUsuario (linhas 70-84)

```
async findByUsuario(usuarioId: string) {
  return this.prisma.avaliacao.findMany({
    where: { usuarioId },
    // ...
  });
}
```

Impacto: Retorna todas as avaliações do usuário sem filtrar por empresa. Um usuário pode ter avaliações em múltiplas empresas e ver todas elas.

Solução:

```
// Controller
@Get('usuario')
findByUsuario(@Request() req) {
  return this.avaliacoesService.findByUsuario(req.user.sub, req.user.empresaId);
}

// Service - Opção 1: Filtrar por empresa
async findByUsuario(usuarioId: string, empresaId: string) {
  return this.prisma.avaliacao.findMany({
    where: {
      usuarioId,
      produto: { empresaId }
    },
    include: {
      produto: {
        select: {
          id: true,
          nome: true,
          imagem: true,
        },
      },
    },
    orderBy: { createdAt: 'desc' },
  });
}

// Opção 2: Retornar todas mas identificar a empresa
// Depende do requisito de negócio
```

Problema 2.4 - remove (linhas 86-102)

```
async remove(id: string, usuarioId: string) {
  const avaliacao = await this.prisma.avaliacao.findUnique({
    where: { id },
  });
  // ...
}
```

Impacto: Não valida se a avaliação pertence a um produto da empresa do usuário.

Solução:

```
// Controller
@Delete('/:id')
remove(@Param('id') id: string, @Request() req) {
  return this.avaliacoesService.remove(id, req.user.sub, req.user.empresaId);
}

// Service
async remove(id: string, usuarioId: string, empresaId: string) {
  const avaliacao = await this.prisma.avaliacao.findUnique({
    where: { id },
    include: { produto: true },
  });

  if (!avaliacao) {
    throw new NotFoundException('Avaliação não encontrada');
  }

  if (avaliacao.usuarioId !== usuarioId) {
    throw new ForbiddenException('Você não pode deletar esta avaliação');
  }

  // Validar se o produto pertence à empresa
  if (avaliacao.produto.empresaId !== empresaId) {
    throw new ForbiddenException('Avaliação não encontrada');
  }

  return this.prisma.avaliacao.delete({
    where: { id },
  });
}
```

3. pedidos.service.ts - 1 vazamento CRÍTICO

Localização: backend/src/pedidos/pedidos.service.ts

Problema 3.1 - findMeusPedidos (linhas 83-118)

```
async findMeusPedidos(usuarioId: string, page: number = 1, limit: number = 10) {
  const [pedidos, total] = await Promise.all([
    this.prisma.pedido.findMany({
      where: { clienteId: usuarioId }, // ✗ SEM FILTRO DE EMPRESASID!
      // ...
    }),
    this.prisma.pedido.count({ where: { clienteId: usuarioId } }),
  ]);
  // ...
}
```

Impacto: CRÍTICO! Um cliente que fez pedidos em múltiplas empresas pode ver TODOS os seus pedidos, independente de qual empresa está acessando.

Exemplo de Exploit:

1. Cliente faz pedido na Pizza Express (empresa A)
2. Cliente faz pedido na Burger King (empresa B)
3. Cliente acessa app da Pizza Express
4. Cliente vê pedidos de ambas empresas (vazamento!)

Solução:

```
// Controller já passa empresaId, mas service não usa
@Get('meus')
findMeusPedidos(
  @Request() req,
  @Query('page') page: string = '1',
  @Query('limit') limit: string = '10',
) {
  return this.pedidosService.findMeusPedidos(
    req.user.sub,
    req.user.empresaId, // Adicionar empresaId
    parseInt(page, 10),
    parseInt(limit, 10),
  );
}

// Service
async findMeusPedidos(
  usuarioId: string,
  empresaId: string, // Adicionar parâmetro
  page: number = 1,
  limit: number = 10
) {
  const skip = (page - 1) * limit;

  const [pedidos, total] = await Promise.all([
    this.prisma.pedido.findMany({
      where: {
        clienteId: usuarioId,
        empresaId // Adicionar filtro
      },
      include: {
        itens: {
          include: {
            produto: {
              select: {
                id: true,
                nome: true,
                imagem: true,
              },
            },
          },
        },
      },
      orderBy: { createdAt: 'desc' },
      skip,
      take: limit,
    }),
    this.prisma.pedido.count({
      where: {
        clienteId: usuarioId,
        empresaId // Adicionar filtro
      }
    })
  ]),
];

return {
  pedidos,
  paginacao: {
    total,
    page,
    limit,
    totalPages: Math.ceil(total / limit),
  },
};
}
```

```
    },  
  };  
}
```

Possível Causa do Erro 500 no Gráfico de Vendas

Analisando as imagens fornecidas, o erro “Erro ao carregar dados de vendas” no dashboard pode ser causado por:

1. **Dados inválidos no banco:** Valores Decimal malformados
2. **Conversão de tipos:** Number(pedido.total) pode falhar se total for null
3. **Datas inválidas:** Problemas com parsing de datas

Recomendação: Adicionar try-catch e validação:


```

async getGraficoVendas(
  empresaId: string,
  periodo: 'dia' | 'semana' | 'mes' = 'dia',
  startDate?: Date,
  endDate?: Date,
) {
  try {
    // código existente...

    const vendas = pedidos.reduce((acc, pedido) => {
      try {
        let chave: string;
        const data = new Date(pedido.createdAt);

        if (groupBy === 'day') {
          chave = data.toISOString().split('T')[0];
        } else if (groupBy === 'week') {
          const inicioSemana = new Date(data);
          inicioSemana.setDate(data.getDate() - data.getDay());
          chave = inicioSemana.toISOString().split('T')[0];
        } else {
          chave = `${data.getFullYear()}-${String(data.getMonth() + 1).padStart(2,
'0')}`;
        }

        if (!acc[chave]) {
          acc[chave] = 0;
        }





        const total = Number(pedido.total);
        if (!isNaN(total) && isFinite(total)) {
          acc[chave] += total;
        }

        return acc;
      } catch (itemError) {
        console.error('Erro processando pedido:', pedido.id, itemError);
        return acc;
      }
    }, {});

    return Object.entries(vendas).map(([data, total]) => ({
      data,
      total: Number(total),
    }));
  } catch (error) {
    console.error('Erro em getGraficoVendas:', error);
    throw new InternalServerErrorException('Erro ao buscar dados de vendas');
  }
}

```

Plano de Ação

1.  Identificar todos os vazamentos (concluído)
2.  Corrigir dashboard.service.ts (2 correções)
3.  Corrigir avaliacaoes.service.ts (4 correções)
4.  Corrigir pedidos.service.ts (1 correção CRÍTICA)

5. ⌚ Adicionar error handling no gráfico de vendas
 6. ⌚ Testar todas as correções
 7. ⌚ Commit atômico das correções
-

Testes Necessários

Após correções, validar:

1. ☒ Dashboard carrega dados apenas da empresa correta
 2. ☒ Avaliações só são criadas em produtos da empresa
 3. ☒ Cliente vê apenas pedidos da empresa atual
 4. ☒ Gráfico de vendas não retorna erro 500
 5. ☒ Produtos não são compartilhados entre empresas
-

Impacto Estimado

- **Segurança:** CRÍTICO - Vazamento de dados entre empresas
- **Performance:** Nenhum impacto negativo
- **Breaking Changes:** Nenhum (correções internas)
- **Riscos:** Baixo - Correções são adições de filtros seguros