How (not) to do robust API Authorization

By Szymon Drosdzol

Self-intro

- My name is Szymon Drosdzol
- I’m a senior security engineer @ 

- We do white box security reviews of web apps
- Contact me at:
  - https://www.linkedin.com/in/szymon-drosdzol/

Agenda

- Introduction
- Simple example
- Architecture anti-patterns
- Good architecture
- Hardening
- QA

Introduction

- Broken access control is number 1 OWASP Top 10
- Found universally in almost every codebase
- There will never be a silver bullet for that class of vulnerabilities
- Therefore, **architecture is everything**

Couple of disclaimers

- All shown vulnerabilities were found in production code
- The impact varied from medium to critical
- All bugs were anonymized
  - No company's names, sorry!
  - All endpoints names converted into a simple project management API
- This presentation tries to be technology-agnostic

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        is_admin = self._user.role == "admin"
        if is_admin:
            if (company_id not in self._user.company_ids):
                raise PermissionDenied()

            return Response(company_service.fetch_company(company_id))
```

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        is_admin = self._user.role == "admin"
        if is_admin:
            if (company_id not in self._user.company_ids):
                raise PermissionDenied()

            return Response(company_service.fetch_company(company_id))
```

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        is_admin = self._user.role == "admin"
        if is_admin:
            if (company_id not in self._user.company_ids):
                raise PermissionDenied()

            return Response(company_service.fetch_company(company_id))
```

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        is_admin = self._user.role == "admin"
        if is_admin:
            if (company_id not in self._user.company_ids):
                raise PermissionDenied()

            return Response(company_service.fetch_company(company_id))
```

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        is_admin = self._user.role == "admin"
        if is_admin:
            if (company_id not in self._user.company_ids):
                raise PermissionDenied()

            return Response(company_service.fetch_company(company_id))
```

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        is_admin = self._user.role == "admin"
        if is_admin:
            if (company_id not in self._user.company_ids):
                raise PermissionDenied()

            return Response(company_service.fetch_company(company_id))
```

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        is_admin = self._user.role == "admin"
        if is_admin:
            if (company_id not in self._user.company_ids):
                raise PermissionDenied()

            return Response(company_service.fetch_company(company_id))
```

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        is_admin = self._user.role == "admin"
        if is_admin:
            if (company_id not in self._user.company_ids):
                raise PermissionDenied()

        return Response(company_service.fetch_company(company_id))
```

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        is_admin = self._user.role == "admin"
        if is_admin:
            if (company_id not in self._user.company_ids):
                raise PermissionDenied()

        return Response(company_service.fetch_company(company_id))
```

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        is_admin = self._user.role == "admin"
        if is_admin:
            if (company_id not in self._user.company_ids):
                raise PermissionDenied()

        return Response(company_service.fetch_company(company_id))
```

Is this a good architecture?

```
class CompanyController(APIView):

    def get(self, req: Request) -> Response:

        company_id = req.data.get("company_id", None)

        return Response(company_service.fetch_company(company_id))
```

Systemic Problems

- Does not provide any framework to developers
- Far from self-documenting
- Becomes unreadable quickly
- Hard to maintain
- Hard (or impossible) to catch bugs with SAST
- Fail-Open

What are the other options?

And what can go wrong?

Seemingly innocent code

```
def jsonify(to_dump: Any, include_pii: bool = False) -> Response:
    # irrelevant code
        # LAST LINE OF DEFENCE TO NOT JUMP CUSTOMERS
        if hasattr(to_dump, 'customer_uuid'):
            if not to_dump.customer_uuid == g.user.customer_uuid:
                raise exceptions.AuthorizationError()
```

Seemingly innocent code

```
def jsonify(to_dump: Any, include_pii: bool = False) -> Response:
    # irrelevant code
        # LAST LINE OF DEFENCE TO NOT JUMP CUSTOMERS
        if hasattr(to_dump, 'customer_uuid'):
            if not to_dump.customer_uuid == g.user.customer_uuid:
                raise exceptions.AuthorizationError()
```

Seemingly innocent code

```
def jsonify(to_dump: Any, include_pii: bool = False) -> Response:
    # irrelevant code
        # LAST LINE OF DEFENCE TO NOT JUMP CUSTOMERS
        if hasattr(to_dump, 'customer_uuid'):
            if not to_dump.customer_uuid == g.user.customer_uuid:
                raise exceptions.AuthorizationError()
```

Seemingly innocent code

```
def jsonify(to_dump: Any, include_pii: bool = False) -> Response:
    # irrelevant code
        # LAST LINE OF DEFENCE TO NOT JUMP CUSTOMERS
        if hasattr(to_dump, 'customer_uuid'):
            if not to_dump.customer_uuid == g.user.customer_uuid:
                raise exceptions.AuthorizationError()
```

Seemingly innocent code

```
class CustomerOwned:
    #irrelevant code

    @declared_attr
    def customer_uuid(cls):
        return Column(
            UUID(as_uuid=True),
            ForeignKey('customers.uuid'),
            nullable=False,
        )

class UserModel(CustomerOwned):
    # properties
```

Seemingly innocent code

```
class CustomerOwned:
    #irrelevant code

    @declared_attr
    def customer_uuid(cls):
        return Column(
            UUID(as_uuid=True),
            ForeignKey('customers.uuid'),
            nullable=False,
        )

class UserModel(CustomerOwned):
    # properties

class IdentityModel:
    # properties
```

Email leak

```
GET /api/v1/identities?limit=1000 HTTP/2
```

Response:

```
{
    "records": [
        {
            "uuid": "d29255a1-5e61-4384-83d6-204d810223ad",
            "created_at": "2021-06-25T18:47:21.023150Z",
            "updated_at": "2021-06-25T18:47:21.023150Z",
            "email": "anon@acme.com",
            "email_verified": true,
        },
        {
            "uuid": "d29255a1-5e61-4384-83d6-204d810223ad",
            "created_at": "2021-06-25T18:47:21.023150Z",
            "updated_at": "2021-06-25T18:47:21.023150Z",
            "email": "anon@acme2.com",
            "email_verified": true,
        },
        …
    ]
```

Session Takeover

```
GET /api/v1/identities?limit=1000&relationships=tokens HTTP/2

Response:
{
    "records": [
        {
            "uuid": “d29255a1-5e61-4384-83d6-204d810223ad",
            "created_at": "2021-06-25T18:47:21.023150Z",
            "updated_at": "2021-06-25T18:47:21.023150Z",
            "email": "anon@acme.com",
            "email_verified": true,
            "account_status": null,
            "tokens": [
                {
                    "client_id": “REDACTED_BY_DOYENSEC”,
                    "token_type": "Bearer",
                    "refresh_token": null,
                    "revoked": false,
                    "issued_at": 1624994285,
                    "expires_in": 604800,
                    "uuid": "REDACTED",
                    "oauth2_identity_uuid": "REDACTED_BY_DOYENSEC"
                    "access_token": “REDACTED_BY_DOYENSEC”
                }
                //snipped for brevity
            ]
        }
    ]
```

Impact

- **One request to take over all sessions in the system**
- Disclosure of all account owners’ emails

Simple Fix?

```
class CustomerOwned:
    #irrelevant code

    @declared_attr
    def customer_uuid(cls):
        return Column(
            UUID(as_uuid=True),
            ForeignKey('customers.uuid'),
            nullable=False,
        )

class UserModel(CustomerOwned):
    # properties

class IdentityModel(CustomerOwned):
    # properties
```

Systemic Problems

- Fail-Open Ownership Check

- Close coupling of API with backend data model
  - Lack of data transport objects (DTOs)
  - Generic mechanisms allowing to “explore” the data model

Fail-Open Mechanisms

```
DOMAINS_ADMIN = Capability(
    'project_write',
    [
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        …
    ],
)
```

Fail-Open Mechanisms

```
DOMAINS_ADMIN = Capability(
    'project_write',
    [
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
          '),



    ],
)
```

Fail-Open Mechanisms

```
DOMAINS_ADMIN = Capability(
    'project_write',
    [
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{u
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/descr
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{u
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/descr
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{u
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/descr
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{u
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/descr
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{u
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/descr
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{u
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/descr
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{u
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/descr
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{u
          '),

    ],
)
```

```
DOMAIN_ADMIN = Capability(
    'project_read',
    [
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}

    ],
)
```

```
USER = Capability(
    'project_read',
    [
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),
        ProtectedRoute('POST', '/v4/projects'),
        ProtectedRoute('PUT', '/v4/projects/{project}'),
        ProtectedRoute('PUT', '/{v}/projects/{project}/describe'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}'),
        ProtectedRoute('DELETE', '/{v}/projects/{project}/{user}'),

    ],
)
```

Impact

- Over 50 vulnerable endpoints
- Mostly mundane, but:
  - Privilege escalation possible
  - Logs stealing possible
  - Domain specific admin actions possible

Systemic Problems

- Forgotten endpoints don’t execute privilege check
- Hard to maintain
- Lack of a solid framework for developers

Old Endpoints

```
class AccountV2List(APIView):
    @required_access(access=LEVEL.ADMIN)
    def get(
        self, request, request_context: RequestContext, account_id,
format=None
    ):
        return
            ProjectV2Service.get_by_account_id(
                request_context, , int(account_id)
            )
        )
```

Old Endpoints

```
class AccountV2List(APIView):
    @required_access(access=LEVEL.ADMIN)
    def get(
        self, request, request_context: RequestContext, account_id,
format=None
    ):
        return
            ProjectV2Service.get_by_account_id(
                request_context, , int(account_id)
            )
        )
```

Old Endpoints

```
class AccountV1List(APIView):
    def get(
        self, request, request_context: RequestContext, account_id,
format=None
    ):
        return
            ProjectV1Service.get_by_account_id(
                request_context, int(account_id)
            )
        )
```

Impact

- Low-privilege users can list all tenant users
  - With emails
- Privilege escalation
  - Edit invitations' roles
- Therefore, a guest can take over the tenant

Systemic problems

- Multiple ways of doing the same thing
- Lack of universal privilege execution mechanism
- Probably missing regular housekeeping

Inconsistent Design

```
class UserView(ProxyView):
    http_method_names: Tuple[str, ...] = ("post",)
    permission_classes: Tuple[BasePermission, ...] = (AllowAny,)

class UserView(APIView):
    permission_classes: Tuple[BasePermission, ...] = (
        IsAuthenticated,    )
    required_permission: str = "retrieve"
```

Inconsistent Design

```
class UserView(ProxyView):
    http_method_names: Tuple[str, ...] = ("post",)
    permission_classes: Tuple[BasePermission, ...] = (AllowAny,)

class UserView(APIView):
    permission_classes: Tuple[BasePermission, ...] = (
        IsAuthenticated,    )
    required_permission: str = "retrieve"
```

Inconsistent Design

```
class UserView(ProxyView):
    http_method_names: Tuple[str, ...] = ("post",)
    permission_classes: Tuple[BasePermission, ...] = (AllowAny,)

class UserView(APIView):
    permission_classes: Tuple[BasePermission, ...] = (
        IsAuthenticated,    )
    required_permission: str = "retrieve"
```

Inconsistent Design

```
class ProjectView(ProxyView):
    permission_classes: Tuple[BasePermission, ...] = (
        IsAuthenticated,
        AllowedRolesPermission,
    )
    allowed_roles: Tuple[str, ...] = ("root",)

class ProjectView(ApiView):
    http_method_names: Tuple[str, ...] = ("get", "post")
    queryset = QuarterComment.objects
    permission_classes: Tuple[BasePermission, ...] =
(IsAuthenticated,)
```

Inconsistent Design

```
class ProjectView(ApiView):
    http_method_names: Tuple[str, ...] = ("get", "post")
    queryset = QuarterComment.objects
    permission_classes: Tuple[BasePermission, ...] =
(IsAuthenticated,)
```

It’s only a matter of time

```
class ReportExport(ProxyView):
    http_method_names: Tuple[str, ...] = ("get",)
    permission_classes: Tuple[BasePermission, ...] = (AllowAny,)

class ReportExport(ApiView):
    http_method_names: Tuple[str, ...] = ("get")
    queryset = QuarterComment.objects
    permission_classes: Tuple[BasePermission, ...] =
(IsAuthenticated,)
```

Impact

- Numerous domain data revealed to low-privilege users
- Possible cross-tenant leaks
  - Impact limited by using GUIDs to identify objects

Systemic Problems

- Multiple ways of doing the same thing
- Unclear privilege model
  - Role-based vs privilege-based
- Inconsistent authorization enforcement location

Path Traversal

```
routes.get('/ip-location', async (req, res, next) => {
  try {
    const response = await req.apiServiceClient.get(
      `/v2/employees/location/${req.query.ip}`
    );
    forwardResponse(res, response);
  } catch (e) {
    next(e);
  }
});
```

Path Traversal

```
# Req.query.ip = '192.168.0.1'

routes.get('/ip-location', async (req, res, next) => {
  try {
    const response = await req.apiServiceClient.get(
      `/v2/employees/location/192.168.0.1`
    );
    forwardResponse(res, response);
  } catch (e) {
    next(e);
  }
});
```

Path Traversal

```
# Req.query.ip = '../'

routes.get('/ip-location', async (req, res, next) => {
  try {
    const response = await req.apiServiceClient.get(
      `/v2/employees/location/../`
    );
    forwardResponse(res, response);
  } catch (e) {
    next(e);
  }
});
```

Path Traversal

```
# Req.query.ip = '../'

routes.get('/ip-location', async (req, res, next) => {
  try {
    const response = await req.apiServiceClient.get(
      `/v2/employees/`
    );
    forwardResponse(res, response);
  } catch (e) {
    next(e);
  }
});
```

Path Traversal

```
GET /ip-location?ip=../
Host: acme.com

[
  {
    "id": "u123456",
    "username": "janedoe",
    "email": "janedoe@example.com",
    "fullName": "Jane Doe",
    "createdAt": "2024-01-15T10:30:00Z",
    "lastLogin": "2025-04-17T14:20:00Z",
    "roles": ["user", "editor"],
    "isActive": true
  },
…
]
```

Path Traversal Impact

- A simple utility endpoint becomes a gateway to any backend path
- Cross-tenant user disclosure

Systemic Problems

- Lack of incoming data validation
- Lack of defined outgoing data models
- Lack of authorization in the internal service

Parameter Pollution

```
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id')
    note = request.values.get('note')
    project_service.add_task(project_id, note)
```

Parameter Pollution

```
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id')
    note = request.values.get('note')
    project_service.add_task(project_id, note)
```

Parameter Pollution

```
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id')
    note = request.values.get('note')
    project_service.add_task(project_id, note)
```

Parameter Pollution

```
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id')
    note = request.values.get('note')
    project_service.add_task(project_id, note)
```

Parameter Pollution

```
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id')
    note = request.values.get('note')
    project_service.add_task(project_id, note)
```

Parameter Pollution

```
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id')
    note = request.values.get('note')
    project_service.add_task(project_id, note)

def api_authorized(f=None, **wrapped_kwargs):
    for param in can_manage:
    obj_id = get_field(request, param, None)
    if not obj_id:
        abort(404)

    if not current_user.can_manage_obj(obj_id):
        return False
```

Parameter Pollution

```
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id')
    note = request.values.get('note')
    project_service.add_task(project_id, note)

def api_authorized(f=None, **wrapped_kwargs):
    for param in can_manage:
    obj_id = get_field(request, param, None)
    if not obj_id:
        abort(404)

    if not current_user.can_manage_obj(obj_id):
        return False
```

Parameter Pollution

```python
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id')
    note = request.values.get('note')
    project_service.add_task(project_id, note)

def api_authorized(f=None, **wrapped_kwargs):
    for param in can_manage:
    obj_id = get_field(request, param, None)
    if not obj_id:
        abort(404)

    if not current_user.can_manage_obj(obj_id):
        return False
```

Parameter Pollution

```
def get_request_values(request):
    """
    Unify all request `args` and `form` fields.
    Note: flask `request.values` only gets args on `"GET"` requests
    and not `.form`
    Args:
        request (flask.Request): flask request object.
Returns:
        dict: All request fields.
    """
    params = {
        k:v for k, v in
        list(request.args.items()) +
        list(request.values.items()) +
        list(request.form.items())

    }
    return params
```

Parameter Pollution

```
def get_request_values(request):
    """
    Unify all request `args` and `form` fields.
    Note: flask `request.values` only gets args on `"GET"` requests
    and not `.form`
    Args:
        request (flask.Request): flask request object.
Returns:
        dict: All request fields.
    """
    params = {
        k:v for k, v in
        list(request.args.items()) +
        list(request.values.items()) +
        list(request.form.items())

    }
    return params
```

Parameter Pollution

```
def get_request_values(request):
    """
    Unify all request `args` and `form` fields.
    Note: flask `request.values` only gets args on `"GET"` requests
    and not `.form`
    Args:
        request (flask.Request): flask request object.
Returns:
        dict: All request fields.
    """
    params = {
        k:v for k, v in
        list(request.args.items()) +
        list(request.values.items()) +
        list(request.form.items())

    }
    return params
```

Parameter Pollution

```
POST /task/add HTTP/1.1
Host: acme.com
Connection: close
Content-Type: application/jaon
Content-Length: 0

{
        "project_id": 456,
        "note": “lorem ipsum”
}
```

Parameter Pollution

```
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id') 456
    note = request.values.get('note')
    project_service.add_task(project_id, note)

def api_authorized(f=None, **wrapped_kwargs):
    for param in can_manage:
    obj_id = get_field(request, param, None). 456
    if not obj_id:
        abort(404)

    if not current_user.can_manage_obj(obj_id): 456
        return False
```

Parameter Pollution

```
POST /task/add?project_id=123 HTTP/1.1
Host: acme.com
Connection: close
Content-Type: application/jaon
Content-Length: 0

{
        "project_id": 456,
        "note": “h@ck3d"
}
```

Parameter Pollution

```
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id') 123
    note = request.values.get('note')
    project_service.add_task(project_id, note)

def api_authorized(f=None, **wrapped_kwargs):
    for param in can_manage:
    obj_id = get_field(request, param, None). 456
    if not obj_id:
        abort(404)

    if not current_user.can_manage_obj(obj_id): 456
        return False
```

Parameter Pollution

```
@api_v1.route('/task/add', methods=['POST'])
@api_authenticated(require_at_least=AccessLevel.USER)
@params_filter(required=['project_id', 'note'])
@api_authorized(can_manage=['project_id'])
def api_v1_task_add():
    project_id = request.values.get('project_id') 123
    note = request.values.get('note')
    project_service.add_task(project_id, note)

def api_authorized(f=None, **wrapped_kwargs):
    for param in can_manage:
    obj_id = get_field(request, param, None). 456
    if not obj_id:
        abort(404)

    if not current_user.can_manage_obj(obj_id): 456
        return False
```

Systemic Problems

- Overly generic parameter retrieval
- Multiple ways of accessing the object_id
- Predictable object IDs

Good Architecture

- Self-documenting
- Fail-Closed
- Single way of doing things
- Ensures that authorization rules are global
- Does not require devs to remember stuff
- Tenant isolation at data layer

Hardening

- Only return necessary data
- Validate input data. Enforce data types
- Enforce authorization in all (micro)services
- Return invitation codes, API keys only on creation
- Do not mix super-admin services with regular clients

What to do with limited time?

- Refactor the most impactful endpoints
- Utilize custom rules scanners (I love Semgrep)

```
rules:
- id: empty_authentication_classes
  patterns:
    - pattern: router.$VERB($NAME, ...)
    - pattern-not: router.$VERB($NAME, ..., auth, ...)
  message: auth is not called for $VERB $NAME
  languages: [javascript]
  severity: WARNING
```

- Utilize the custom rules in CI/CD and create a control to improve over time
- Migrate to GUIDs as opposed to integer ids

What to do with limited time?

- Refactor the most impactful endpoints
- Utilize custom rules scanners (I love Semgrep)

```
rules:
- id: empty_authentication_classes
  patterns:
    - pattern: router.$VERB($NAME, ...)
    - pattern-not: router.$VERB($NAME, ..., auth, ...)
  message: auth is not called for $VERB $NAME
  languages: [javascript]
  severity: WARNING
```

- Utilize the custom rules in CI/CD and create a control to improve over time
- Migrate to GUIDs as opposed to integer ids

What to do with limited time?

- Refactor the most impactful endpoints
- Utilize custom rules scanners (I love Semgrep)

```
rules:
- id: empty_authentication_classes
  patterns:
    - pattern: router.$VERB($NAME, ...)
    - pattern-not: router.$VERB($NAME, ..., auth, ...)
  message: auth is not called for $VERB $NAME
  languages: [javascript]
  severity: WARNING
```

- Utilize the custom rules in CI/CD and create a control to improve over time
- Migrate to GUIDs as opposed to integer ids

Thanks!