BDI port knocking

basic data infrastructure

Topsector Logistiek

Colophon

BDI port knocking

Authors
Joost Diepenmaat
Michiel de Mare
Remco van 't Veer

July 2024

basic data infrastructure

Topsector Logistiek

Contents

Contents

1 Introduction

1.1 Abstract

BDI, or Basic Data Infrastructure, is a framework designed to standardize and streamline the collection, sharing, and use of data across various sectors. In BDI, data owners control access to their data, ensuring protection against unauthorized users. This document explores to what extent port knocking and related technologies can enhance data security in a BDI setting.

1.2 Terminology

- **Allowlisting IP Addresses:** A security measure that allows only specified IP addresses to access a network, system, or application, effectively blocking all other traffic.
- **AWS Security Groups:** Virtual firewalls for Amazon Web Services (AWS) instances, controlling inbound and outbound traffic based on a configurable ruleset.
- **CDN (Content Delivery Network):** A network of distributed servers that deliver web content to users, while applying a range of security measures, including DDoS protection, rate limiting and rules based filtering.
- **Data Owner:** An entity that possesses and has control over specific data.
  Data owners are responsible for defining the terms and conditions under which their data can be accessed and shared with other authorized parties. In iSHARE, this role is also known as an Entitled Party.
- **Delegation Evidence:** Documentation or proof that authority or access rights have been transferred from one entity to another, often used in security and trust management contexts.
- **Guard Component:** A security module or mechanism designed to monitor, control, and protect a system by enforcing access controls and policies.
- **Knock Agent:** A process which is connected to a knock server, receives instructions to allow or deny IP addresses, and apply these to configured backends such as firewalls.
- **Knock Server:** A server which provides the web interface that users log into, and that interacts with an identity provider.
- **Port Knocking:** A security technique where a series of network requests (knocks) on closed ports is used to communicate information, typically to open a port for further access.
- **Service Provider:** An entity that controls access to resources in a way that sharing complies with the iSHARE framework's standards for security, privacy, and governance.
- **Reverse Proxy:** A server that sits between client devices and backend servers, forwarding client requests to the appropriate server and returning the server's response to the clients. It helps with load balancing, security, and caching.
- **VPC (Virtual Private Cloud):** An isolated, configurable virtual network environment within a public cloud, allowing users to provision resources, manage network settings, and establish secure connections.

1.3 Assumptions

In this document, we have made the following assumption: Only API calls over the HTTP/HTTPS protocol should be considered. Other protocols need not be taken into account.

Note that for simplicity we use the term "owner" in the diagram in this document to mean a self hosting party (the data owner is also the service provider). The proposed solutions also apply to situations where a separate service provider is involved, unless stated otherwise.

2 Problem Statement

The iSHARE protocol centers around the concept of "data at the source," meaning that responsibility for securing data access is distributed among various parties. While the protocol outlines authentication and authorization procedures, each service provider is also responsible for safeguarding their data against malicious actors, who may attempt to bypass or overwhelm these procedures. iSHARE is a complicated protocol. The security of the network is dependent on the correct implementation and control measures of individual parties. Can we provide mechanisms to improve the safety of BDI services?

2.1 Current authentication/authorization procedure

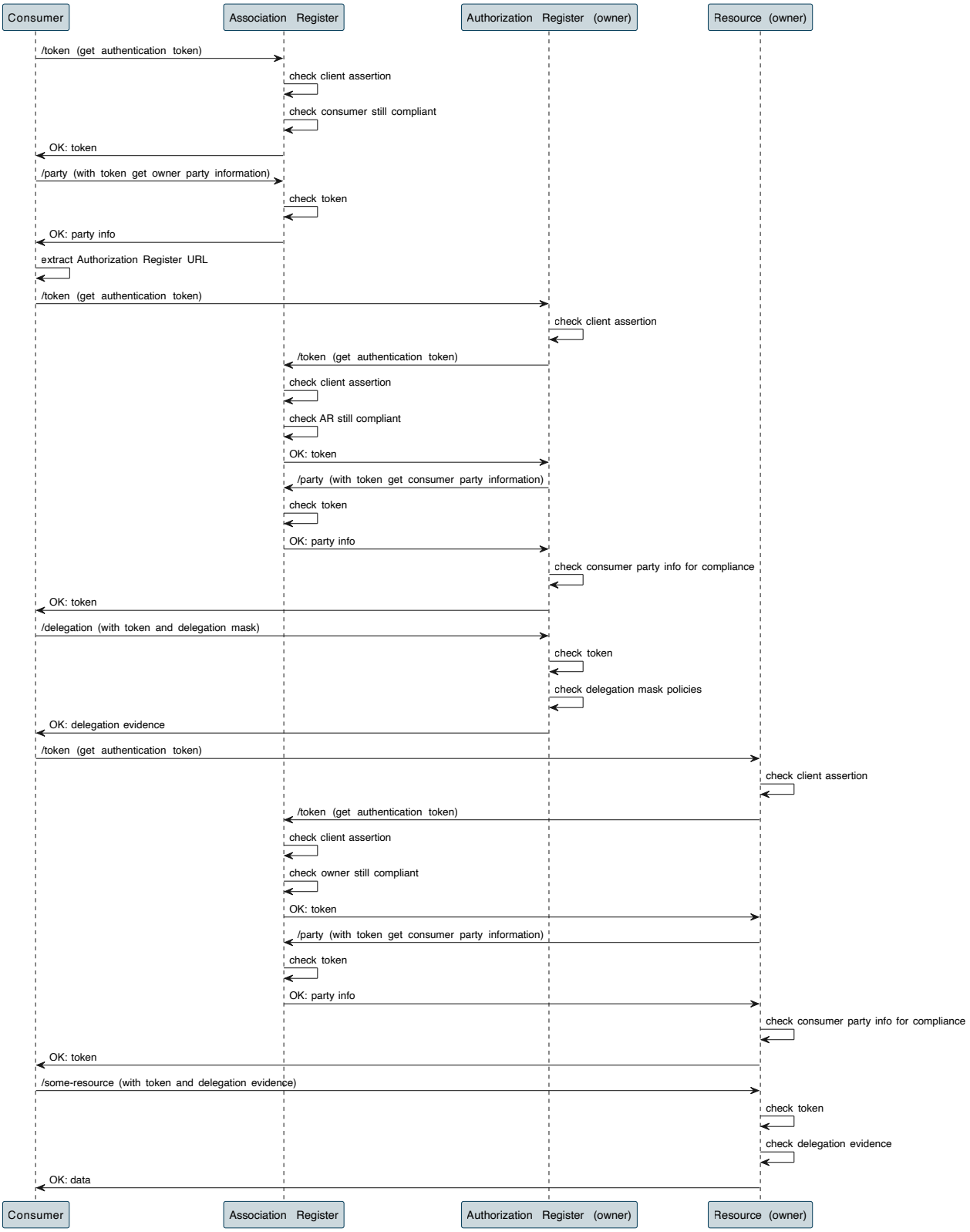
Currently, the sequence of steps to get data from a resource flows as illustrated in figure 1.

2.2 Desired Changes

It is undesirable that the data sources of the data owner are publicly reachable. We propose to enhance security of the protocol by adding a **guard** component that acts as a gatekeeper to the data sources. Access to a data source requires clearance from a gatekeeper after successful authentication.

Changes to the BDI protocols will be considered, but we prefer changes that will not complicate data consumer (client sofware) implementations - since clients will likely outnumber service providers.

Figure 1
Base sequence



3 Gatekeeping and Port Knocking

We want to restrict access to sensitive resources, and we don't want to let those resources handle all the burden of authentication and authorization because of the problems enumerated in the "Security Challenges" section.

3.1 Port Knocking

Port knocking is a security technique where a series of network requests (knocks) on closed ports is used to communicate information, typically to open a port for further access. It is suitable for protocols with persistent connections, such as TCP and SSH. HTTP has stateless connections and is not a suitable protocol for classic port knocking.

3.2 Knock Servers

Another way is the Knock Server approach, as implemented by knockknoc.io. In this setup, the resources to be protected are inaccessible by default. To access a resource, one must first be authenticated with a knock server, which then instructs a Knock Guard to open a specific port for an IP address.

Knockknoc.io is designed primarily for Human to Machine (H2M) interactions. For Machine to Machine (M2M) interactions, the OAuth protocol can be used instead of signing in on a web page.

There are a few limitations with this approach. First, the Knock Server is still vulnerable to attack. Problems with the Knock Server would make all resources unavailable from outside. Secondly, the lease awarded by the Knock Agent is time based. That could be a problem, both when the interval is too short, and when it is too long. Finally, when the port is closed, the client will receive a "connection closed" error instead of a clear error message, which makes it hard to troubleshoot for developers and users of client software.

This approach can be implemented for any component, including the Association Register, APIs of the Service Provider, data (e.g. files on http servers) and the Authorization Register.

However, protecting services that are called from other services (such as the Association Register) using a Knock Server will lead to a cascade of requests.

4 Practical considerations

4.1 Introduction

Before starting the implementation of a guard component, the following points will need to be resolved.

4.2 Conceptual problems

4.2.1 Stable IP-addresses

Is the IP address of the client stable enough for solutions that involve allowlisting of IP-addresses such as Port Knocking? In M2M environments, multiple IP-address changes may occur each day. To what extent will failures to be allowlisted caused by IP-address changes cause problems?

4.2.2 Whack-a-mole

Are we just moving the problem around? An attacker wanting to perform a DoS attack will now target the BDI port knock (or Authorization Register) implementation, instead of the API of the resource owner. Do we have to put those behind a CDN?

4.2.3 Simple Security versus Complex HTTP

A Knock Server approach involves allowlisting IP-addresses after a HTTP call. HTTP is a complex protocol, and therefore HTTP-servers are invariably quite complex. However, from a security point of view, it is recommended for the process granting access to be as simple as possible. These two considerations are at odds with each other.

Similarly, an implementation based on a http reverse proxy such as HAProxy will remain vulnerable to Zero Day Attacks. With classic Port Knocking, the attack surface will be significantly smaller.

4.3 Implementation problems

4.3.1 Install at every supplier

Similar to the Authorization Register, every Service Provider that wishes to use a guard component will have to install it separately.

4.3.2 Low latency needed

The knocknock.io service is intended for H2M, but we are looking for a M2M solution which may involve much higher peak traffic, and shorter and faster interactions. It needs to be possible to grant or deny access on all supported backends (such as AWS security groups, HAProxy iptables) with fairly low latency (in the order of a second or less).

4.3.3 Block until access granted

When granting access, the initial "knock" call should be blocking until the access has actually been granted and the client's IP-address has been allowlisted, to ensure that the client's next request will not be blocked by the firewall.

4.3.4 IPv6

We should also support IPv6. Extra effort is needed to make sure that allowlisting IP addresses also works with IPv6, not just IPv4.

4.3.5 Another layer to debug

When access to data is denied, there is now another layer to debug. We should create clear error messages in the access logs to debug problems with denied data access.

4.3.6 Various network configurations

The network configurations at the suppliers are going to diverge quite a bit. We will probably encounter the following configurations:

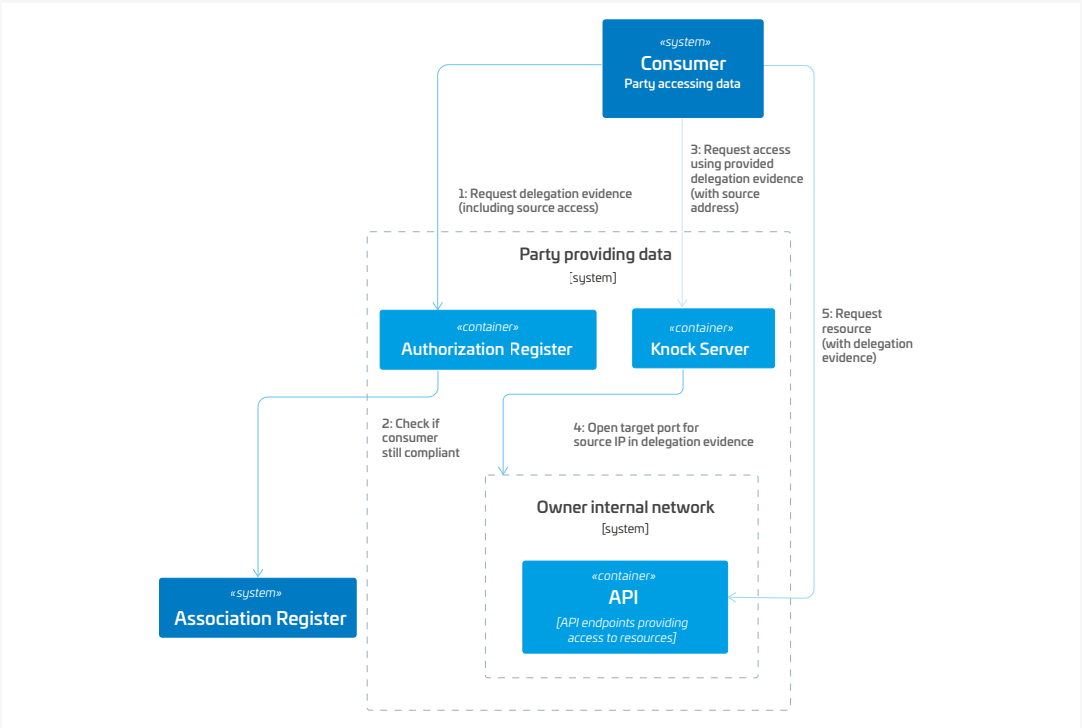
- **Intranets**
  An intranet is not reachable from the internet at all. Some kind of gateway is needed to provide access to services running on an intranet.
- **AWS/Azure private networks (VPC)**
  Provide network isolation at the virtual level. Subnets can be designated as private (without direct internet access) or public (with internet access via an Internet Gateway)
- **Web applications on the internet**

Since Port Knocking involves the network configuration, we will need to support the most common types of network configuration in order to be able to provide some kind of off-the-shelf solution.

5 Solution: Knock Server

The knock server opens a firewall/cloud security group for the duration of the call; the knock server will validate the delegation evidence and open access for the IP address therein for the lifespan of the delegation evidence. A consumer first calls this component, and then to the actual API calls as illustrated in figure 2.

Figure 2
Knockknoc server container



The new sequence of steps to get data from a response flows as illustrated in figure 3.

Pros and cons

This approach has the following advantages and drawbacks.

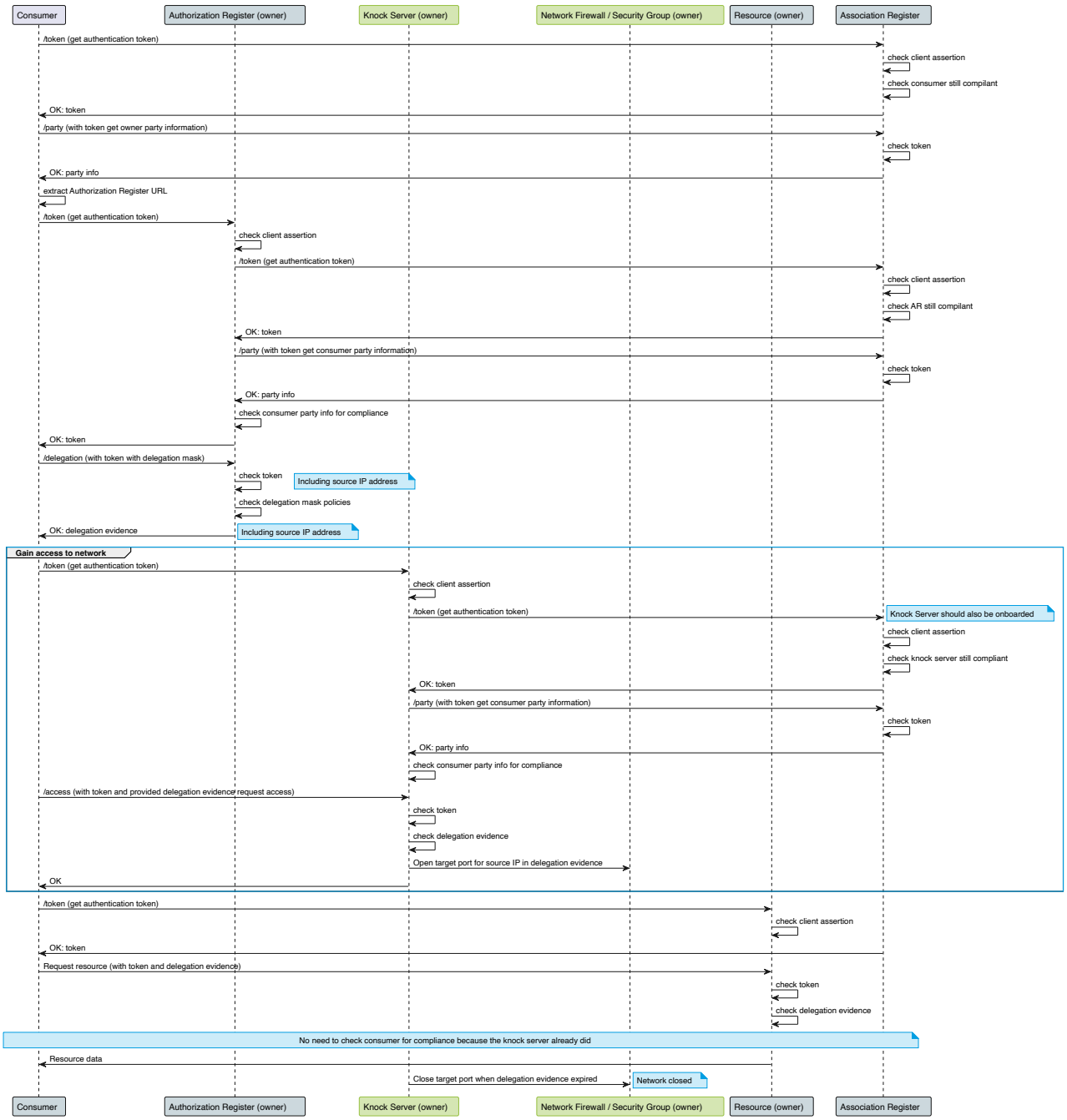
Pros

- Attack surface reduced to knock server, Authorization Register, and (hopefully) battle tested firewall/cloud security groups.
- Not limited to HTTP traffic.

Cons

- More steps for a consumer to access data.
- Possibly hard to determine when the port can be closed.
- Consumer must provide Delegation Evidence to gain access - this prevents the "Minimum implementation" (see also https://dev.ishare.eu/serviceconsumer/getting-started.html#scenario-1-minimum-implementation) authorization flow where the Service Provider is responsible for collecting delegation evidence.

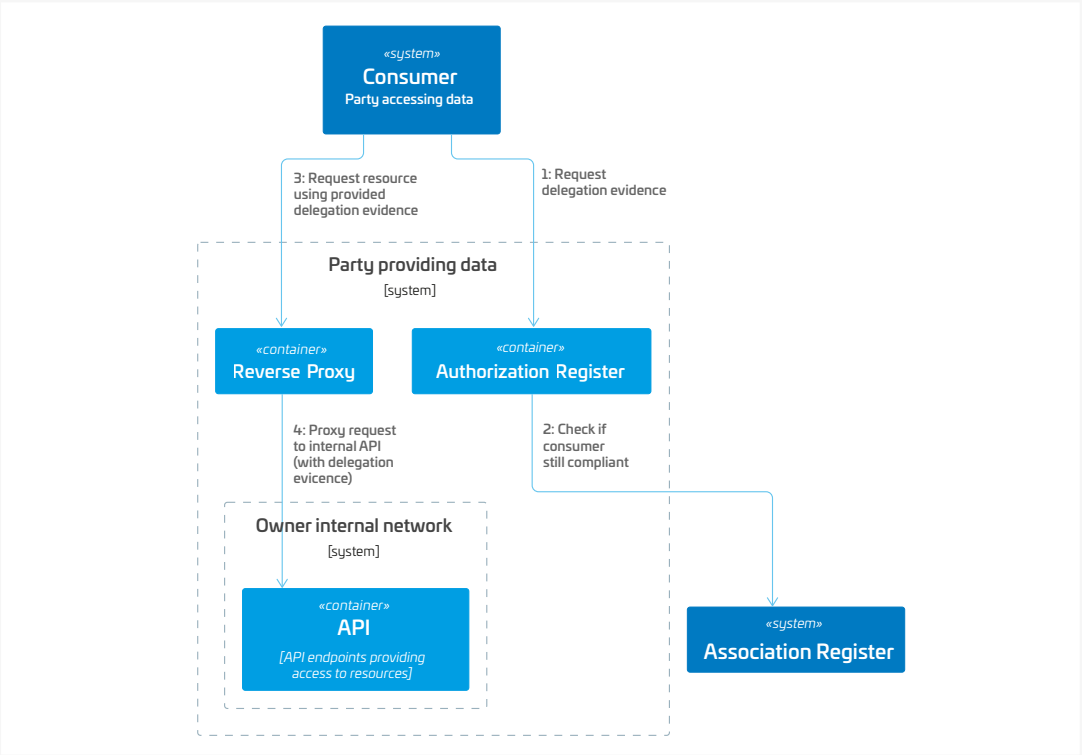
Figure 3
Knockknoc server sequence



6 Solution: Reverse Proxy

Instead of opening the network to clients, force clients to access a single endpoint, and HTTP reverse proxy, and perform any security checks there. A reverse proxy which is able to validate the delegation evidence and provide direct access to the resource within the owner's (or service provider) network as illustrated in figure 4. This solution corresponds to the "BDI Authentication Processor" (see https://bdinetwork.org/resources/glossary/).

Figure 4
Reverse proxy container



In this scenario the proxy is responsible for authenticating the client, and can also do preliminary checks of the provided delegation evidence (if present). It can check that the subject of the delegation evidence is the client, that the issuer is a supported associaton registry, that all parties are members of the expected association etc. However, the downstream resource API will still need to check that the provided delegation evidence actually applies to the requested resource. The proxy does not match the request to the delegation evidence, it only checks that the provided delegation evidence is trustworthy. If no delegation evidence is provided, the requested resource has to ensure authentication, for instance, by requesting delegation evidence from the relevant authorization registry.

If the client does not provide a delegation evidence, as in "the minimum implementation" (see also https://dev.ishare.eu/service-consumer/gettingstarted.html#scenario-1-minimum-implementation) flow, the proxy can either deny access, or it can allow access, in which case the downstream resource is responsible for requesting the appropriate delegation evidence.

The new sequence of steps to get data from a response is as illustrated in figure 5.

Pros and cons

This approach has the following advantages and drawbacks.

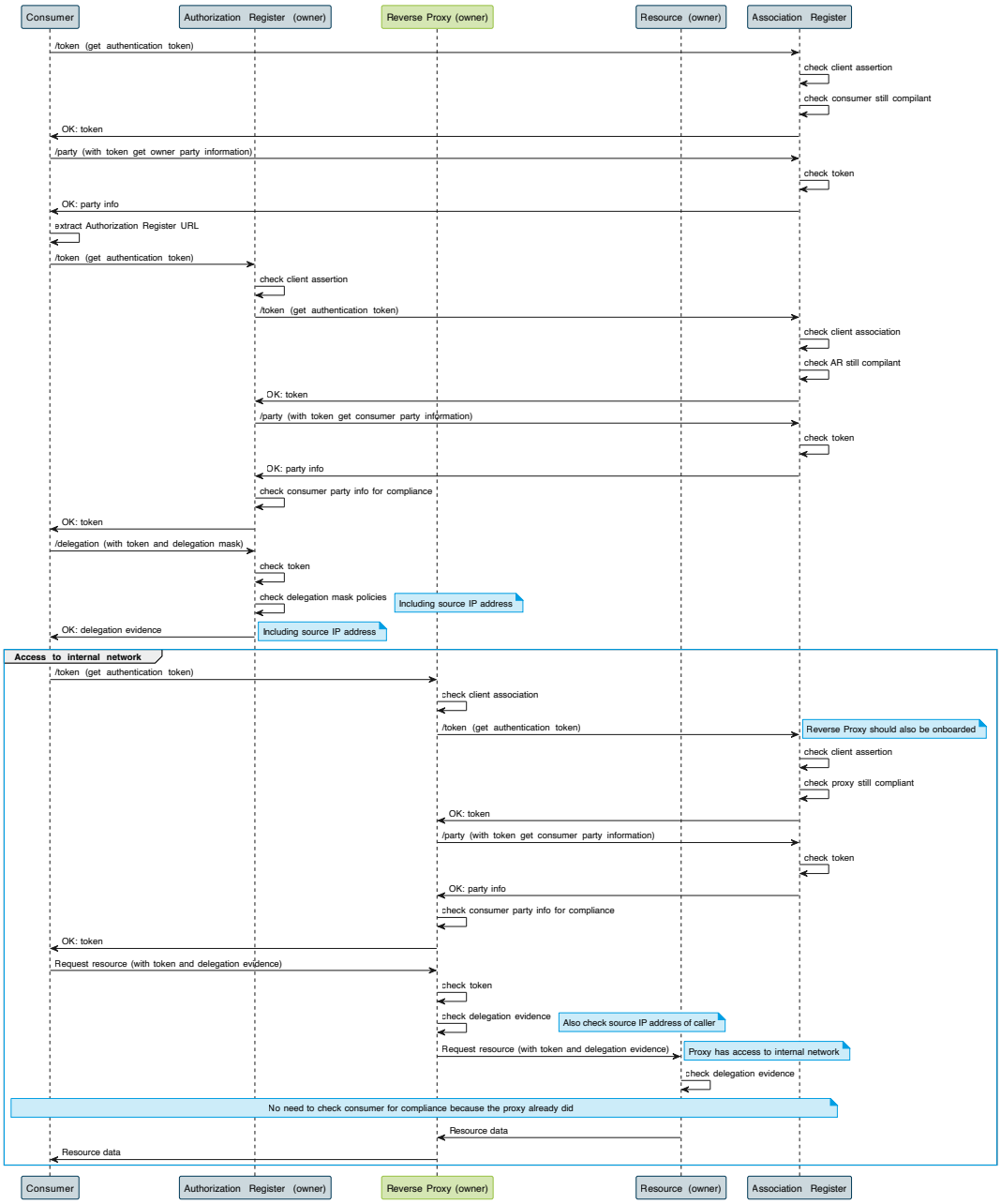
Pros

- Transparent access for consumers - can be implemented without any changes to the BDI protocols.
- No time limits on access; no reauthentication.
- Services behind the reverse proxy may choose to skip authentication and parts of the authorization checks since this has been handled already by the reverse proxy.
- Since all traffic passes through the reverse proxy, it is a natural place for a SSL endpoint, so that servers upstream do not have to use SSL.

Cons

- Since all access passes through the reverse proxy, adding or removing a resource means changing its configuration.
- Reverse proxies are complex pieces of software, and are therefore vulnerable to Zero Day Attacks.

Figure 5
Reverse proxy sequence



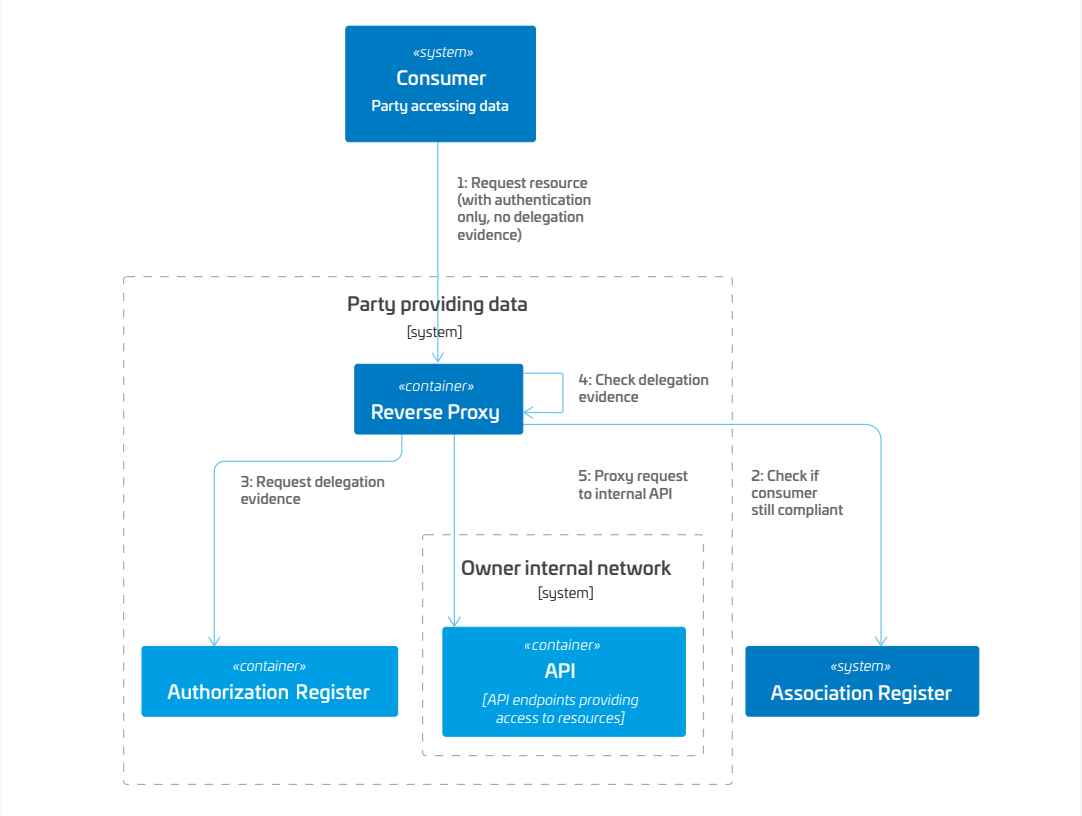
7 Solution: Full authorization proxy

A proxy can be implemented that would fetch and evaluate delegation evidence needed for the consumer's request, if missing. For this to work, the proxy must be able to map an incoming request to a delegation mask and can ensure that an incoming request is fully authenticated and authorized given the access token and delegation evidence.

This means the service implementation can hand off most of the access control to the proxy, but the proxy needs to know more about the service's API in order to map and validate requests against delegation evidence.

See figure 6 for an overview.

Figure 6
Full authorization proxy container



Pros and cons

This approach has the following drawbacks and advantages.

Pros

- Simpler service implementation that doesn't need to communicate with other iSHARE services (all done in the proxy).
- Allows for all M2M authentication flows described in iSHARE.

Cons

- The proxy needs to be configured to map incoming requests to delegation masks and delegation evidence. This increases complexity of the proxy; it becomes more difficult to prove the proxy implementation, including its configuration, is secure.

8 Solution: Generic Service Provider

A further extension of the "Full authorization" variant, that includes a translation of incoming iSHARE-style HTTP calls to whatever internal API the downstream service implements. This scenario is described in "The journey to an iShare Generic Service Provider" by René Kint & Wouter Sieburgh for Cargonaut.

In addition to the Pros and Cons of the "Full authorization" variant, we have:

Pros and cons

This approach has the following advantages and drawbacks.

Pros

- Allows for exposing existing "legacy" interfaces through BDI without changes to the legecy system itself.

Cons

- Translation of incoming HTTP calls to unspecified legacy systems is an unlimited problem. This will probably involves custom code and/or plugging in various existing integrations of uncertain quality. It may be worth splitting the translation layer into its own "translation proxy", so as to not increase the complexity of the authorization proxy even further.

9 Alternative Approaches

9.1 Client certificates

Clients already have been supplied certificates in a certificate chain. Requiring a valid client certificate for both the Authorization Service and the actual service would keep out all kinds of bad actors, at a cost of some added complexity for the client.

This added complexity is quite modest and code samples can be provided. Checking the client certificate is built-in functionality of the http server, load balancer or reverse proxy. Cloudflare also supports checking client certificates - this is called Mutual TLS (mTLS).

The main disadvantage of this approach is that it only works if all clients have been issued a client certificate from a trusted Certificate Authority (CA). Whether clients participating in BDI/iSHARE have already been granted such a certificate, or whether it is feasible to distribute client certificates to all participants is subject to further research.

9.2 Extend Authorization Register

Write an extension for the Authorization Register (AR), so that it opens the firewall for given source IP address as soon as access has been granted. See Appendix B for a detailed proposal.

A complication arises in that this approach burdens the AR with technical knowledge of third-party services. The iSHARE approach assumes that an AR is chosen by a data owner and is not necessarily managed by the same party as the service being protected. Multiple ARs can exist for different owners/customers, who may then manage the same service provider.

10 Conclusion

10.1 Summary of Findings

We have found that there are a number of effective ways to increase security in an iSHARE environment. Adding a guard component, either in the form of a Knock Server, or in the form of a reverse proxy, would harden BDI configurations against malicious actors.

Furthermore, it is possible and feasible to extend the iSHARE Delegation Evidence to include the IP address of the client, as described in more detail in Appendix B.

10.2 Recommendations

The *Reverse Proxy/Authentication Proxy* setup is the best candidate for improving security and simplifying the implementation of a resource API. It hides a lot of complexity for the service provider and is a clear verifiable boundary between the internet and the internal network.

All proposed solutions will also need a load balancer and possibly other measures to mitigate Denial of Server attacks but those are out of scope of this document.

10.3 Future Work

We recommend investigating whether client certificates can be deployed to all participants. If all potential users can be issued a client certificate by an approved Certificate Authority (CA), having a CDN verify the certificate could be a useful addition to the authentication and authorization protocols of iSHARE. While certificate management has its challenges, if feasible, this approach would be a relatively simple solution.

To implement a Knock Guard, it's important to identify all backends that need support (e.g., AWS, Azure, HAProxy, various firewalls). For each backend, research the procedures for adding or removing a allowlisted IP address and determine the associated latency.

Additionally, investigate knockknoc.io to determine its possible role in BDI/iSHARE. Can it be a suitable candidate despite its focus on H2M? Or should we develop a solution focused on M2M?

Appendix A

Security Challenges

A solution involving a gatekeeper can help prevent or mitigate a number of security challenge listed here below:

A.1 Denial of Service (DoS) Attack

A Denial of Service (DoS) attack is a malicious attempt to disrupt the normal functioning of a targeted server, service, or network by overwhelming it with a flood of Internet traffic. The primary goal of a DoS attack is to make the target resource unavailable to its intended users.

There are many kinds of DoS attacks. Preventing bad actors from accessing the servers is a common DoS mitigation strategy. allowlisting IP-addresses is an example of that.

A.2 Brute Force Attacks

In the context of web APIs, a brute force attack is a method used by attackers to gain unauthorized access to the API by systematically trying a large number of possible credentials or keys until they find the correct one. The aim is to exploit the authentication mechanism of the API by guessing the correct username and password combination, API key, or other credentials.

A.3 Scans and Automated Attacks

Scans are automated processes used by attackers to probe web APIs for vulnerabilities. The goal is to map out the API surface, identify endpoints, and find security weaknesses. Common scanning techniques include:

1. **Endpoint Discovery:**
   Automated tools enumerate all accessible endpoints by systematically trying common paths and analyzing API documentation, if available.
2. **Parameter Fuzzing:**
   Sending random or specially crafted data to API parameters to identify how the API handles unexpected input, which may reveal vulnerabilities like SQL injection, XSS, or buffer overflows.
3. **Security Vulnerability Scanning:**
   Using tools like OWASP ZAP or Burp Suite to automatically scan for known security vulnerabilities, such as insecure authentication mechanisms, improper error handling, and outdated software versions.
4. **Header Inspection:**
   Checking HTTP headers for misconfigurations or sensitive information leakage.

Automated attacks involve the use of scripts and tools to exploit the vulnerabilities identified during scanning. These attacks can be wide-ranging and target various aspects of the API:

1. **SQL Injection:**
   Injecting malicious SQL code through API parameters to manipulate the database.
2. **Command Injection:**
   Sending malicious commands through API inputs that are executed by the server.
3. **Data Exfiltration:**
   Automated scripts that exploit vulnerabilities to extract sensitive data from the API.
4. **Session Hijacking:**
   Exploiting session management flaws to take over user sessions and impersonate legitimate users.
5. **API Abuse:**
   Using legitimate functionality of the API in an unintended manner to achieve malicious goals, such as scraping data or bypassing rate limits.

A.4 Zero Day Attack

A zero day attack is a critical security flaw, usually in an operating system or application server, which often allows full access to the system. It exploits a previously unknown vulnerability in software or hardware, occurring before the developer has been able to release a patch or fix.

A.5 Patch Frequency

Ensuring frequent application of security patches can be quite a challenge. Doing so successfully will shrink the attack surface of the application, and therefore its vulnerability to automated attacks. However, patch application involves testing and downtime, and consumes a lot of work for the responsible IT teams.

A.6 Uptime Guarantees

Application providers may have provided uptime guarantees or Service Level Agreements (SLA). Security incidents may affect the uptime.

Appendix B

Extending iSHARE Delegation Mask/Evidence

By extending the shape of the *Delegation Mask/Evidence* `target` to include the source IP address the consume will be using, it is possible to ensure this source will have firewall/reverse proxy access to the resource. The added snippet can look like this:

```
"accessSourceAddress": {
 "IPv4": "123.456.789.012",
 "IPv6": "2001:0db8:85a3:0000:0000:8a2e:0370:7334"
}
```

The given addresses should also support ranges and their notation, like `123.456/16` and `2001:0db8:85a3/48` etc.

Note that at least a IPv4 or IPv6 should be provided and that the *Service Provider* should support any of the given options and a proper response should be designed for when a *Service Provider* does not support the given address type.

Current shape of a *Delegation Mask* (see also https://dev.ishareworks.org/delegation/delegation-request.html):

```
{
 "delegationRequest": {
  "policyIssuer": "EU.EORI.NL123456789",
  "target": {
   "accessSubject": "EU.EORI.NL987654321"
  },
  "policySets": [ "object" ]
 },
 "delegation_path": [ "string" ],
 "previous_steps": [ "string" ]
}
```

Proposed shape after extending it:

```
{
 "delegationRequest": {
  "policyIssuer": "EU.EORI.NL123456789",
  "target": {
   "accessSubject": "EU.EORI.NL987654321",
   "accessSourceAddress": {
     "IPv4": "123.456.789.012",
     "IPv6": "2001:0db8:85a3:0000:0000:8a2e:0370:7334"
    }
   },
   "policySets": [ "object" ]
 },
 "delegation_path": [ "string" ],
 "previous_steps": [ "string" ]
}
```

Appendix B

Current shape of a *Delegation Evidence* (see also https://ishare-3.gitbook.io/ishare- developer-portal/reference/delegation-evidence):

```
{
 "delegationEvidence": {
  "notBefore": 1504683475,
  "nonOnOrAfter": 1504683775,
  "policyIssuer": "EU.EORI.NL123456789",
  "target": {
   "accessSubject": "EU.EORI.NL987654321"
  },
  "policySets": [ "object" ]
 }
}
```

Proposed shape after extending it:

```
{
 "delegationEvidence": {
 "notBefore": 1504683475,
 "nonOnOrAfter": 1504683775,
 "policyIssuer": "EU.EORI.NL123456789",
 "target": {
  "accessSubject": "EU.EORI.NL987654321"
  "accessSourceAddress": {
    "IPv4": "123.456.789.012",
    "IPv6": "2001:0db8:85a3:0000:0000:8a2e:0370:7334"
  }
 },
 "policySets": [ "object" ]
}
}
```

Topsector Logistiek
Ezelsveldlaan 59
2611 RV Delft
+31 15 251 65 65
www.topsectorlogistiek.nl

Topsector Logistiek