June 2021

Improved Security and Reliability in the DNS Provider Discovery Mechanism of Domain Connect Protocol

Navneet Goel

Aditya Gopalakrishnan

Improved Security and Reliability in the DNS Provider Discovery Mechanism of Domain Connect Protocol

ABSTRACT

Domain Connect is a publicly available standard that enables DNS Providers to provide a mechanism for Service Providers to place DNS records on the domain, thus freeing the customer from having to manually set the records. There however exist some security and reliability challenges arising from the current Domain Connect specifications which Service Providers might want to protect themselves from. Specifically the *_domainconnect* TXT record, in theory, can be compromised to point to a server controlled by a bad actor; the protocol does not provide a facility to shut down DNS Providers known to have downtime or other issues; and the specification also doesn't enforce URL fields from the settings call to be on the HTTPS scheme. This disclosure describes an allow-list mechanism that mitigates the above-described security and reliability challenges. A wildcard (or regular-expression) check is conducted on the initial server URL returned from following the *_domainconnect* TXT record for a given domain, e.g., to check the host name in the URL, etc. Subsequent wildcard checks also validate fields that are returned in response to the settings call.

KEYWORDS

- Domain Connect specification
- Domain name system (DNS)
- DNS Provider
- Service Provider
- DNS records
- Wildcard check
- Regex check
- DNS Provider discovery

BACKGROUND

An internet domain name, e.g. example.com, makes it easy for customers to access websites without having to memorize and enter numeric IP addresses for those websites. Customers, such as individuals and businesses, purchase domain names from domain registrars. The domain name system (DNS) is the phonebook of the internet, mapping domain names to their IP addresses by looking up DNS records maintained at name servers. A customer can hire one or more Service Providers to provide various services, e.g. email, documents storage, word processing, advertisement serving etc. on their domain.

A DNS Provider (not to be confused with Service Provider) authoritatively hosts the DNS records such as A, NS, CNAME, MX, TXT etc. [1]. For example, the A record stores the actual IP address associated with the domain. The NS record shows the nameservers currently used by the domain. The CNAME record is a type of resource record in the Domain Name System (DNS) that maps one domain name to another (the canonical name). MX refers to mail servers used by the domain. A TXT record (short for text record) is a type of resource record in the Domain name System (DNS) used to provide the ability to associate arbitrary text with a host or other name, such as human readable information.

Domain Connect is a standard that enables DNS Providers to provide a mechanism for Service Providers to place DNS records on the domain, thus freeing the customer from having to manually set the records [2]. The Service Provider might typically use heuristics or rules to guess the DNS Provider for the domain they serve. The Domain Connect specification also enables the Service Provider to discover the DNS Provider for a domain automatically by following the public TXT record on the *_domainconnect* subdomain to get the address of a settings server. The

Service Provider can then fetch the settings of the DNS Provider for the domain using an HTTP REST API call. The settings have a number of fields in the form of a JSON object e.g.

```
{
"providerId": "xyzdomains.com",
"providerName": "XYZ Domains",
"providerDisplayName": "XYZ Domains",
"urlSyncUX": "https://domainconnect.xyzdomains.com",
"urlAsyncUX": "https://domainconnect.xyzdomains.com",
"urlAPI": "https://api.domainconnect.xyzdomains.com",
"width": 750,
"height": 750,
"urlControlPanel": "https://domaincontrolpanel.xyzdomains.com/?domain=%domain%",
"nameServers": ["ns01.xyzdomainsdns.com", "ns02.xyzdomainsdns.com"]
}
```

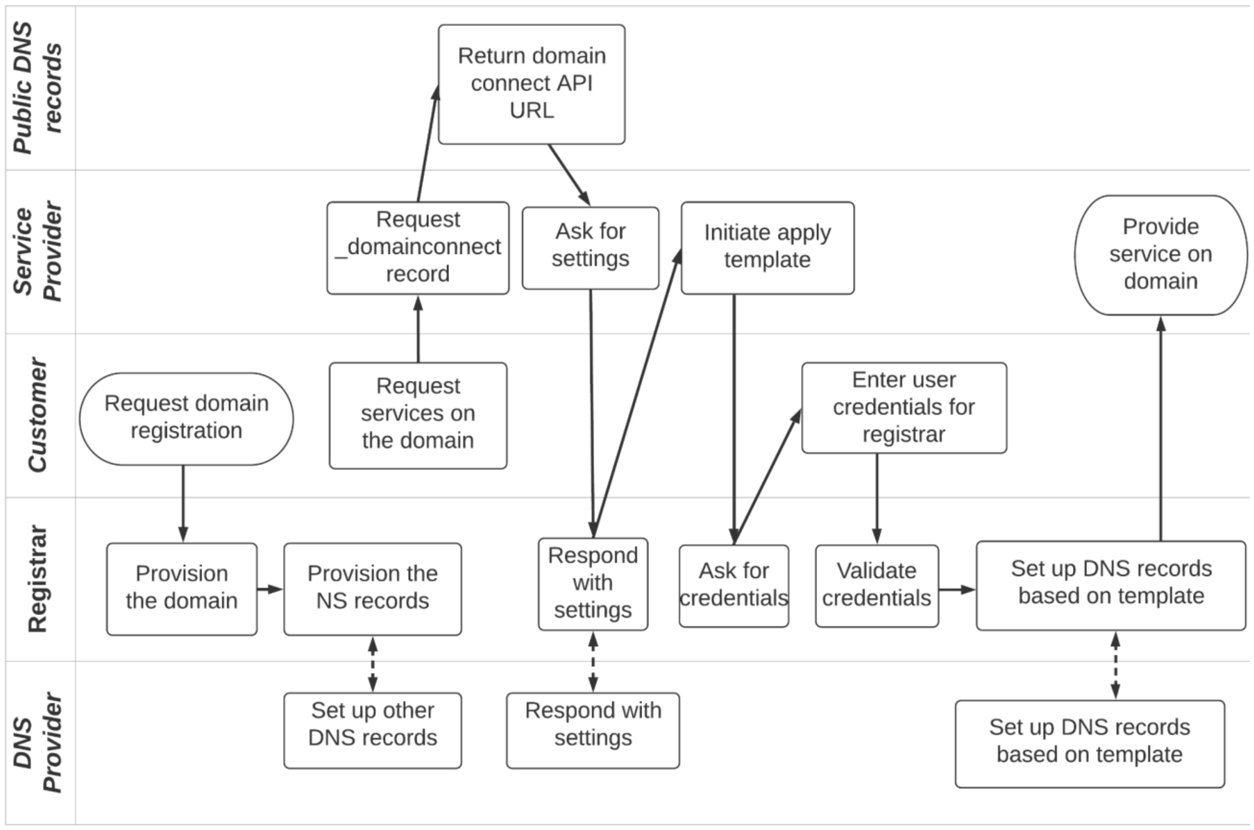

Fig. 1: Synchronous flow with DNS Provider discovery

Fig. 1 illustrates an example workflow (known as synchronous flow) of DNS Provider discovery by a Service Provider using the Domain Connect specification. Upon a customer requesting a domain registration, the registrar provisions the domain and the nameservers. The nameservers point to the DNS Provider. The customer signs up for services with the Service Provider. The Service Provider enables services for the customer by requesting them for permission to apply a Domain Connect template to update or add to the DNS records. The customer authenticates themselves on the DNS Provider website and authorizes the Service Provider to apply the Domain Connect template. Once the DNS records are added or updated successfully, the Service Provider can provide services to the customer.

PROBLEMS

This works in the general case but there are certain challenges pertaining to security and reliability which Service Providers might want to protect themselves from. These include:

- **Problem 1:** The *_domainconnect* TXT record can be compromised to point to a bad actor's server. Subsequent settings call on the URL returned from *_domainconnect* record can then also be compromised in such cases. As per the specification, the URL in the *urlSyncUX* parameter from settings response is actually expected to load the DNS Provider’s user interface (UI). A bad actor however can use this to show an arbitrary UI to the Service Provider's customers.
- **Problem 2:** The protocol doesn't provide a mechanism to shut down DNS Providers by Service Providers where DNS Providers are known to have downtime or other issues.
- **Problem 3:** The specification doesn't enforce or recommend URL scheme validation from the settings call response e.g. checks to enforce URL fields e.g. *urlSyncUX*, *urlAsyncUX*, *urlApi*, *urlControlPanel* etc. are hosted on the HTTPS scheme.

Each of these lead to specific scenarios, described in detail below with examples.

Problem 1: Compromising DNS Provider discovery using a corrupted TXT record

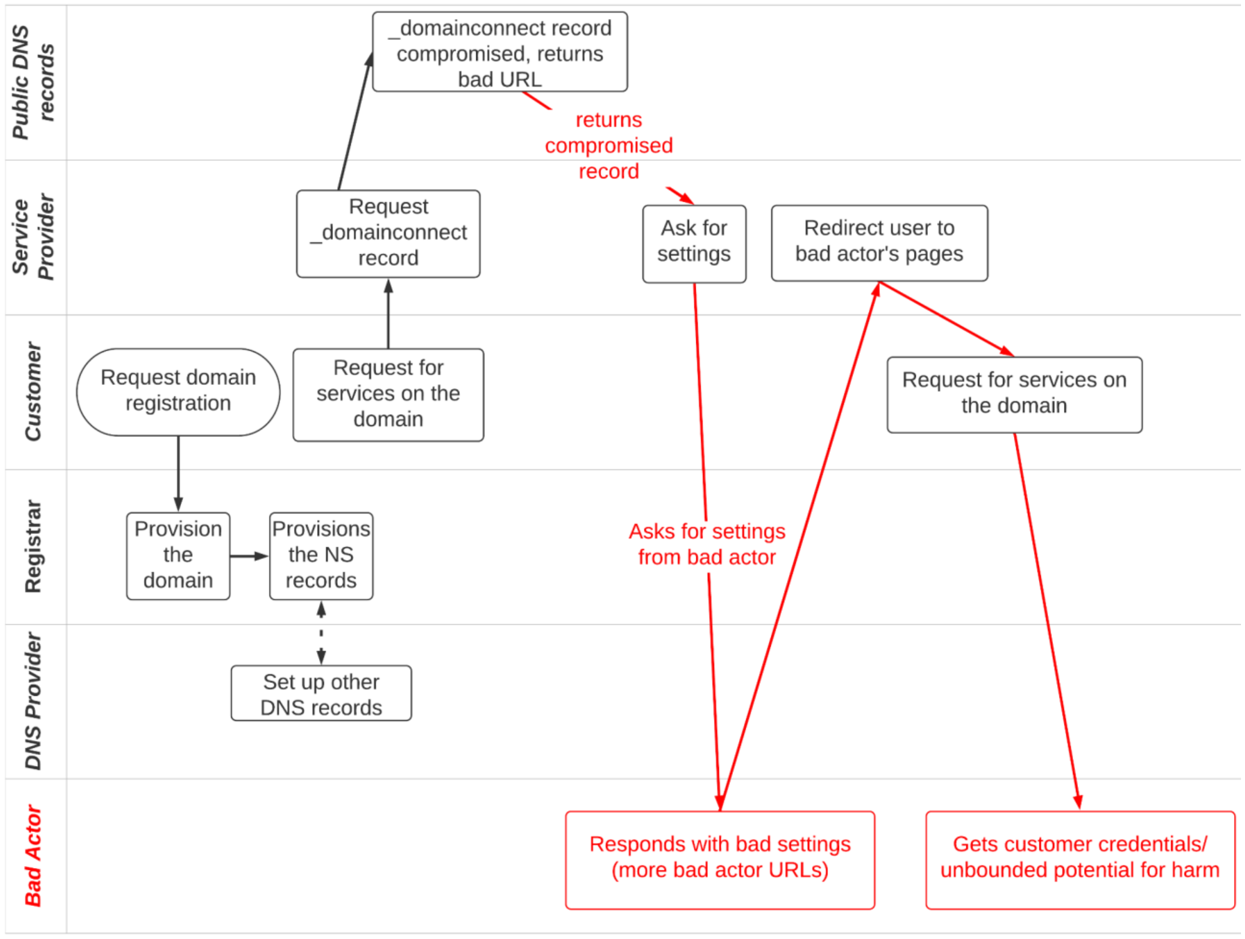
Public DNS records
Service Provider
Customer
Registrar
DNS Provider
Bad Actor
_domainconnect record compromised, returns bad URL
returns compromised record
Request _domainconnect record
Ask for settings
Redirect user to bad actor's pages
Request domain registration
Request for services on the domain
Request for services on the domain
Provision the domain
Provisions the NS records
Asks for settings from bad actor
Set up other DNS records
Responds with bad settings (more bad actor URLs)
Gets customer credentials/ unbounded potential for harm

Fig. 2: Compromising DNS Provider discovery using a corrupted TXT record

Fig. 2 illustrates an example attack mode, e.g. when a bad actor corrupts the *_domainconnect* record on the domain to compromise DNS Provider discovery settings requested by a Service Provider. In such a case, as illustrated, the bad actor can gain the login credentials of the customer etc. with the potential to cause substantial harm.

Example: An example is provided below that illustrates this problem:

1. Assume that the domain of the customer is “mydomain.com”
2. Assume that the *_domainconnect* record has been compromised to contain the hostname “api.badactor.com”, unknown to the customer.
3. A Service Provider providing services to “mydomain.com” would query the *_domainconnect* record in point 2 to get “api.badactor.com”
4. The Service Provider would then proceed to make a settings call for mydomain.com by calling the URL *https://api.badactor.com/v2/mydomain.com/settings*
5. This would then contain bad URLs for integrating with Domain connect like so

```
{
"providerId": "badactor.com",
"providerName": "Bad actor",
"providerDisplayName": "Bad actor",
"urlSyncUX": "https://domainconnect.badactor.com",
"urlAsyncUX": "https://domainconnect.badactor.com",
"urlAPI": "https://api.domainconnect.badactor.com",
"width": 750,
"height": 750,
"urlControlPanel": "https://domaincontrolpanel.badactor.com/?domain=mydomain.com",
"nameServers": ["ns01.badactor.com", "ns02.badactor.com"]
}
```

6. When the Service Provider redirects to any of the URLs as part of the Domain Connect protocol - the customer is now being redirected to bad actor’s URLs where they can suffer loss of login credentials or PII or other possible attacks.

It is true that the customer’s domain is already in a compromised state at the beginning of the flow. But these URLs might compromise aspects of the customer not yet compromised.

Problem 2: An unknown DNS Provider or a DNS Provider suffering an outage/downtime

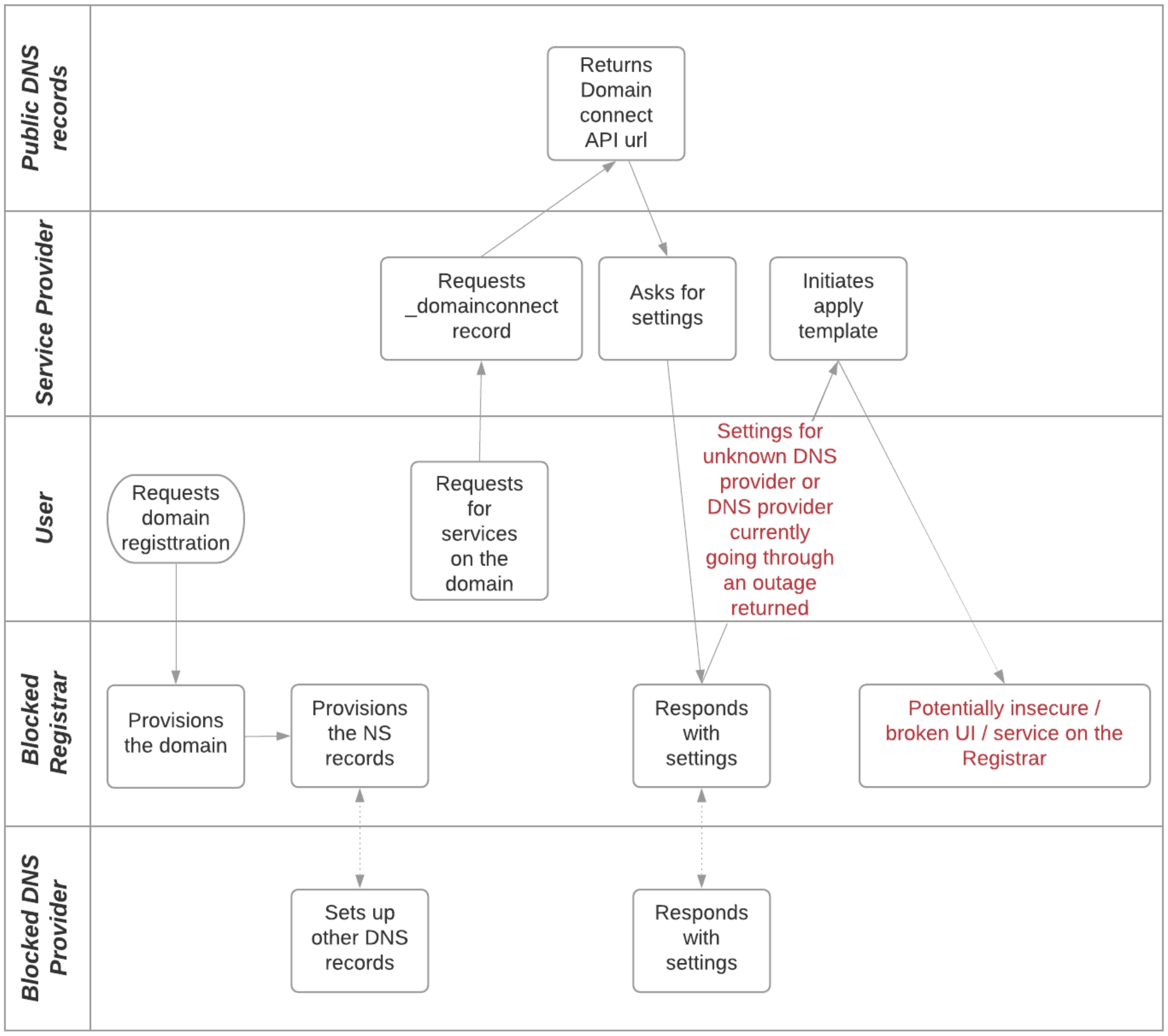
Public DNS records
Service Provider
User
Blocked Registrar
Blocked DNS Provider
Returns Domain connect API url
Requests _domainconnect record
Asks for settings
Initiates apply template
Requests domain registtration
Requests for services on the domain
Settings for unknown DNS provider or DNS provider currently going through an outage returned
Provisions the domain
Provisions the NS records
Responds with settings
Potentially insecure / broken UI / service on the Registrar
Sets up other DNS records
Responds with settings

Fig. 3: A DNS Provider suffering an outage / downtime

Fig. 3 illustrates an example of how a DNS Provider going through downtime / service outage can result in an unreliable experience for the Service Provider. The specification currently has no recommendation for these scenarios. Furthermore, Service Providers might not want to integrate with DNS Providers unknown to them for reliability or business reasons, while still leveraging the benefits of the Domain Connect DNS Provider discovery specification.

Example: An example is provided below that illustrates this problem:

1. Assume that the domain of the customer is “mydomain.com”
2. Assume that the *_domainconnect* record correctly points to the good DNS Provider “MyAwesomeDNSProvider” with the URL “api.myawesomednsprovider.com
3. A Service Provider providing services to “mydomain.com” would query the *_domainconnect* record in point 2 to get “api.myawesomednsprovider.com”
4. The Service Provider would then proceed to make a settings call for mydomain.com by calling the URL *https://api.myawesomednsprovider.com/v2/mydomain.com/settings*
5. This would then contain URLs for integrating with Domain connect like so

```
{
"providerId": "myawesomednsprovider.com",
"providerName": "MyAwesomeDNSProvider",
"providerDisplayName": "MyAwesomeDNSProvider",
"urlSyncUX": "https://domainconnect.myawesomednsprovider.com",
"urlAsyncUX": "https://domainconnect.myawesomednsprovider.com",
"urlAPI": "https://api.domainconnect.myawesomednsprovider.com",
"width": 750,
"height": 750,
"urlControlPanel":
"https://domaincontrolpanel.myawesomednsprovider.com/?domain=mydomain.com",
"nameServers": ["ns01.myawesomednsprovider.com",
"ns02.myawesomednsprovider.com"]
}
```

6. However assume that MyAwesomeDNSProvider is having a known outage which the Service Provider might want to guard against
7. Since there is no good way to detect MyAwesomeDNSProvider as per the current specification, they would continue to integrate with the MyAwesomeDNSProvider and the customer would see a failure on the UI, which could have been avoided.

Problem 3: DNS Provider uses HTTP instead of HTTPS URLs in settings

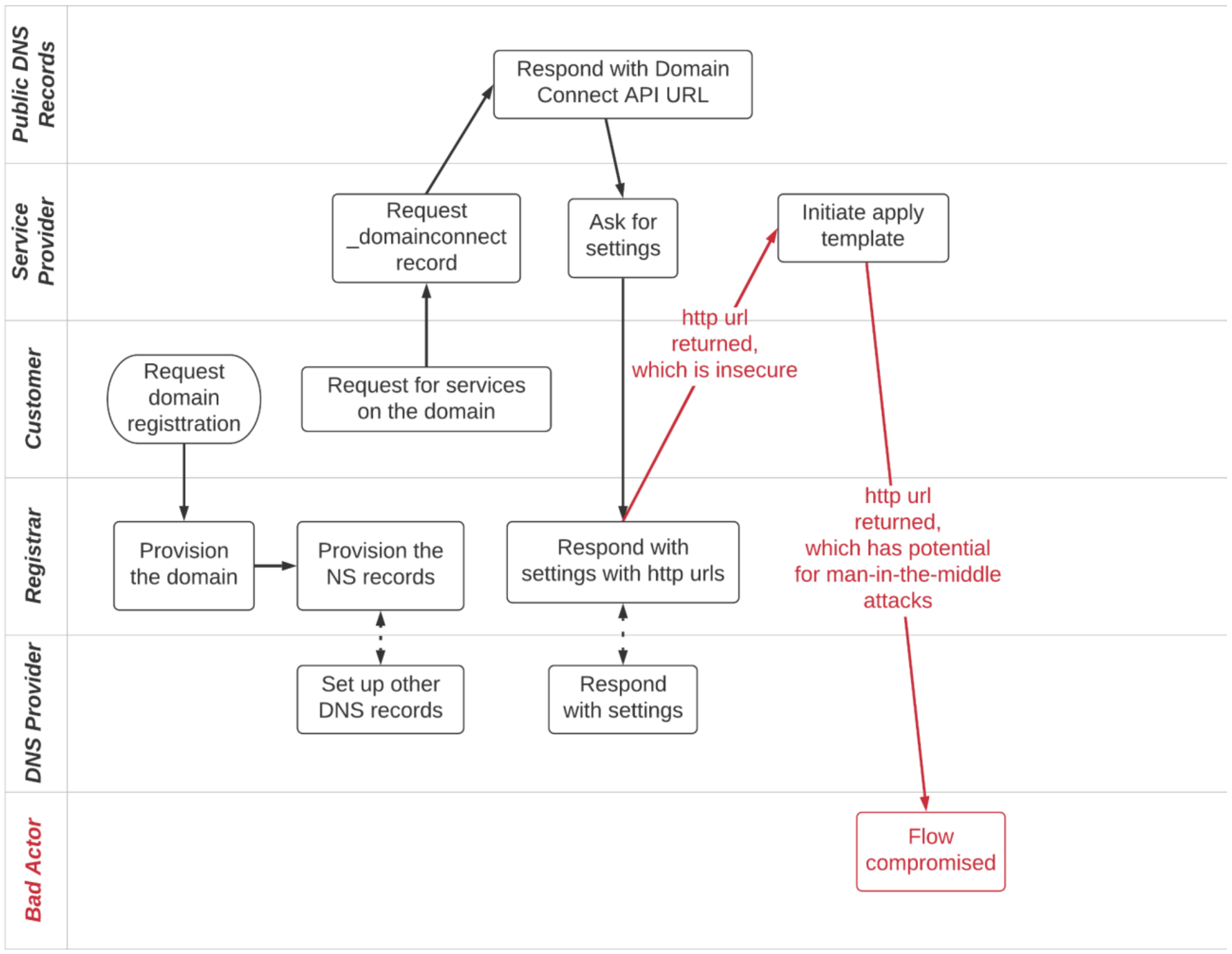
Public DNS Records
Service Provider
Customer
Registrar
DNS Provider
Bad Actor
Respond with Domain Connect API URL
Request _domainconnect record
Ask for settings
Initiate apply template
http url returned, which is insecure
Request domain registtration
Request for services on the domain
http url returned, which has potential for man-in-the-middle attacks
Provision the domain
Provision the NS records
Respond with settings with http urls
Set up other DNS records
Respond with settings
Flow compromised

Fig. 4: Illustration of an existing security issue if the DNS Provider uses HTTP URLs as opposed to HTTPS

Fig. 4 illustrates an existing security issue in the traditional synchronous flow that arises if the DNS Provider uses the HTTP protocol instead of the HTTPS protocol. The use of the less secure HTTP protocol leaves open the possibility of a man-in-the-middle attack which eventually will compromise the overall flow for the user.

Example: An example is provided below that illustrates this problem:

1. Assume that the domain of the customer is “mydomain.com”
2. Assume that the *_domainconnect* record contains the hostname “api.gooddnsprovider.com” for “mydomain.com”
3. A Service Provider providing services to “mydomain.com” would query the *_domainconnect* record in point 2 to get “api.gooddnsprovider.com”
4. The Service Provider would then proceed to make a settings call for “mydomain.com” by calling the URL *https://api.gooddnsprovider.com/v2/mydomain.com/settings*
5. This response might look like this

```
{
"providerId": “gooddnsproviderid”,
"providerName": "gooddnsprovider name",
"providerDisplayName": "gooddnsprovider display name",
"urlSyncUX": "http://domainconnect.gooddnsprovider.com",
"urlAsyncUX": "http://domainconnect.gooddnsprovider.com",
"urlAPI": "http://api.domainconnect.gooddnsprovider.com",
"width": 750,
"height": 750,
"urlControlPanel":
"http://domaincontrolpanel.gooddnsprovider.com/?domain=mydomain.com",
"nameServers": ["ns01.gooddnsprovider.com", "ns02.gooddnsprovider.com"]
}
```

6. As shown in the above response, all the URLs returned are using *HTTP* scheme instead of *HTTPS*. Now when a Service Provider initiates any call e.g. an apply template call using one of the above URLs, the outgoing call will be insecure as it is hosted on *HTTP* scheme instead of *HTTPS* and thus is prone to man-in-the-middle attack which eventually will compromise the overall flow for the user.

SOLUTION

This disclosure describes an allow-list mechanism that mitigates the above described security and reliability challenges. A wildcard (or regular-expression) check is conducted on the initial URL returned from following the *_domainconnect* TXT record. Subsequent wildcard checks also verify the fields such as: *providerId, providerName, providerDisplayName, urlSyncUX, urlAsyncUX*, *urlApi, urlControlPanel* etc. which are returned in response to the settings call. The URLs are additionally verified to be HTTPS, not HTTP.

These checks can be achieved in the following two steps:

1. Getting the pattern for each of the above parameters from the DNS Providers in advance. The pattern type can be one of the pattern types SUBSTRING, STARTSWITH, ENDSWITH, EXACT or any other regular expression based construct.
2. Validating each of these patterns against actual fields which are returned in response to the real time settings JSON call.

A DNS Provider is considered secure only if each of the above checks pass. Furthermore, if a DNS Provider is identified as failing, default fallbacks can be agreed upon with the DNS Provider, thereby mitigating the reliability related challenges. Such fallbacks can be e.g. URLs for the settings *urlSyncUX, urlAsyncUX, urlApi, urlControlPanel* etc. Or in other cases, the Service Provider can turn down traffic to the DNS provider which is suffering any outage or any other issue.

Solution for problem 1: Hardening against a compromised _domainconnect record

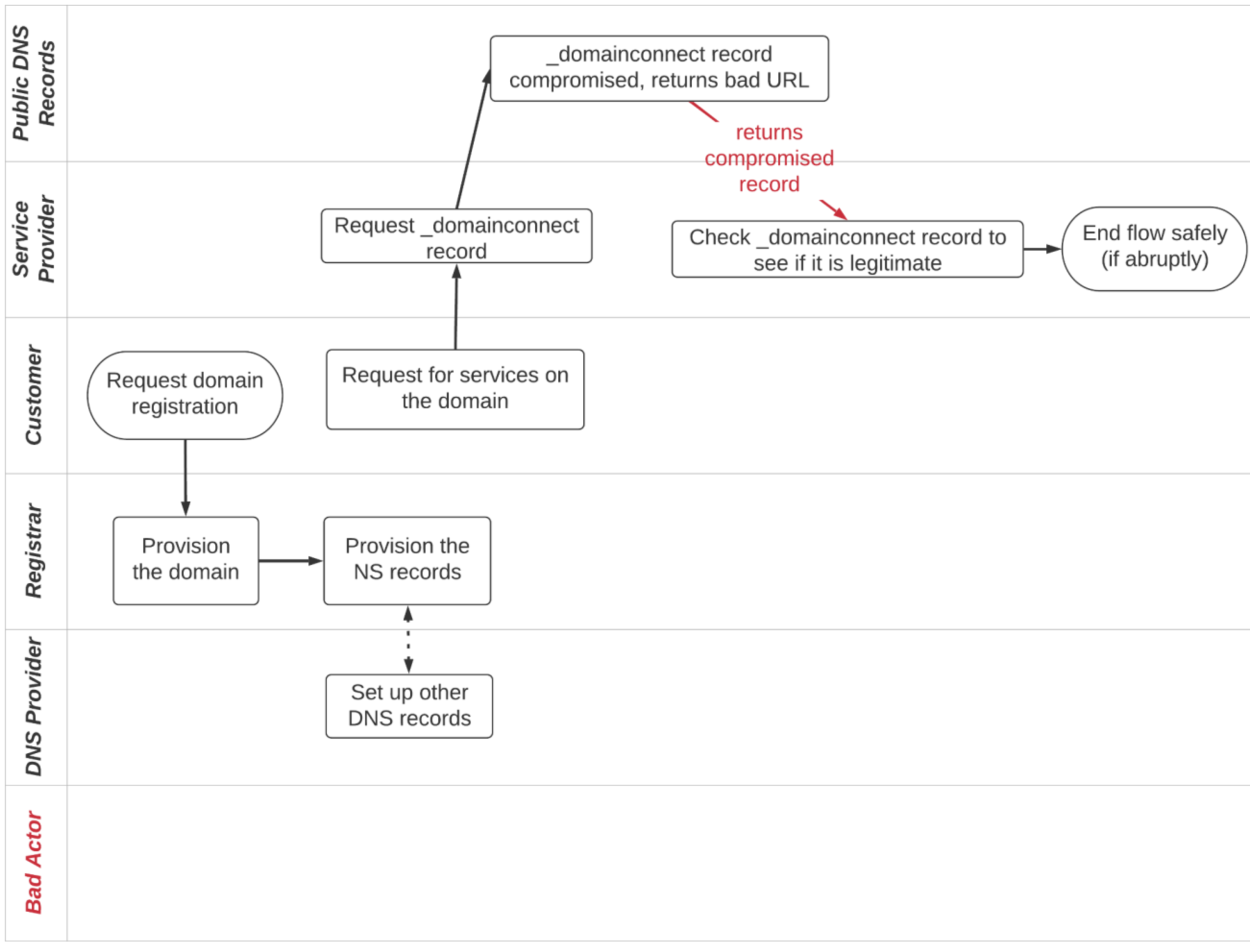
Public DNS Records
Service Provider
Customer
Registrar
DNS Provider
Bad Actor
_domainconnect record compromised, returns bad URL
returns compromised record
Request _domainconnect record
Check _domainconnect record to see if it is legitimate
End flow safely (if abruptly)
Request domain registration
Request for services on the domain
Provision the domain
Provision the NS records
Set up other DNS records

Fig. 5: Hardening against a compromised _*domainconnect* record

Fig. 5 illustrates hardening against a compromised _*domainconnect* record, per the techniques of this disclosure. In contrast to Fig. 2, a Service Provider can detect, using the described techniques, a compromised _*domainconnect* record and immediately end the flow, shunting out the bad actor.

Example: An example is provided below that illustrates the solution.

1. Assume that the domain of the customer is “mydomain.com”

2. Assume that the _*domainconnect* record has been compromised to contain the hostname "api.badactor.com", unknown to the customer.
3. A Service Provider providing services to "mydomain.com" would query the _*domainconnect* record in point 2 to get "api.badactor.com"
4. However, at this very point, the Service Provider would check this URL retrieved against known wildcard patterns of all known good DNS Providers they wish to integrate with e.g. consider the following database on the Service Provider side created with agreement with DNS Providers.

DNS Provider	**Settings URL pattern**	**Settings URL pattern type**
MyAwesomeDNS	"myawesomedns.com"	SUBSTRING
MyAwesomeDNS2	"api.myawesomedns2.com"	EXACT

5. At this point - the Service Provider checks the retrieved settings API server i.e. api.badactor.com against both providers it has chosen to integrate with
   a. *MyAwesomeDNS's* pattern translates to the pattern "**myawesomedns.com**" - which the retrieved URL doesn't match against
   b. *MyAwesomeDNS2's* pattern is an exact match against "*api.myawesomedns2.com*" the retrieved URL also doesn't match against.
6. The Service Provider decides therefore to reject this settings URL and the customer is therefore protected. While such rejection may result in blocking some good DNS providers, it ensures that integrating with bad DNS providers is eliminated. This tradeoff may be preferable than the high costs of integrating with a bad provider.

Solution for problem 2: Hardening against DNS Providers suffering outage/downtime

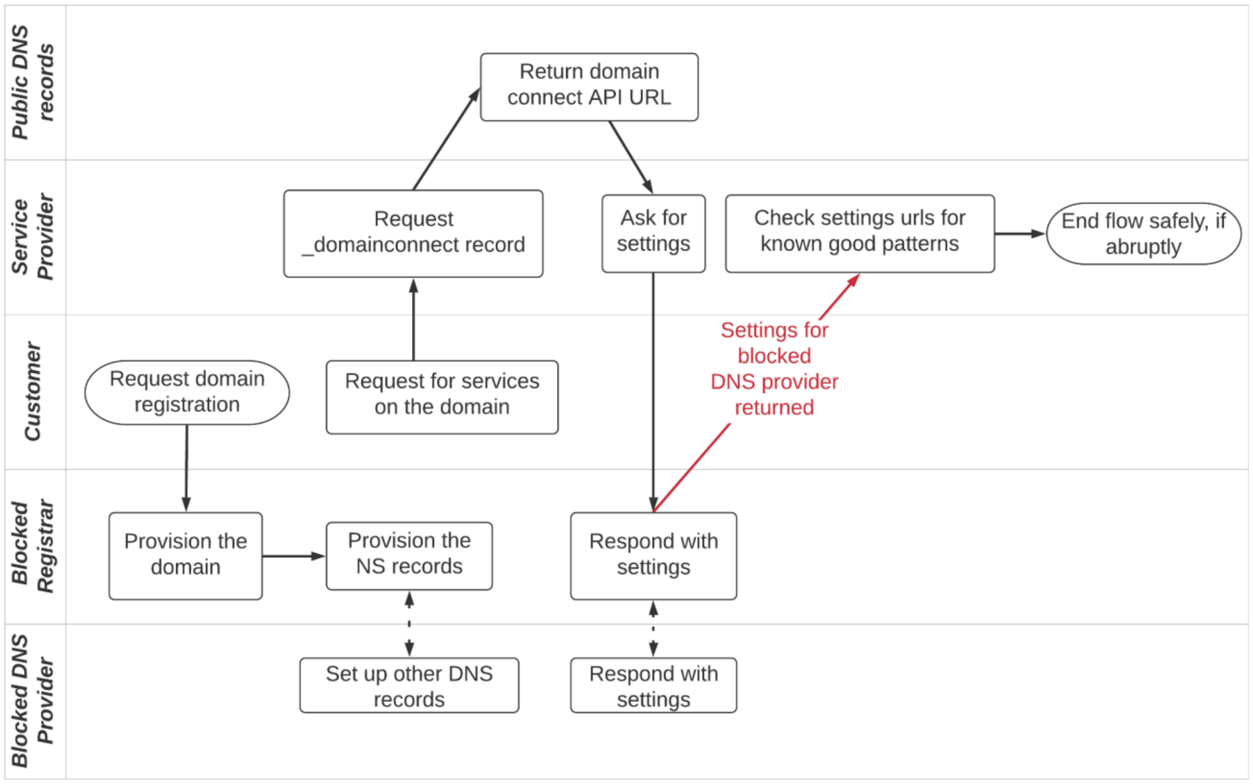
Public DNS records
Service Provider
Customer
Blocked Registrar
Blocked DNS Provider
Return domain connect API URL
Request _domainconnect record
Ask for settings
Check settings urls for known good patterns
End flow safely, if abruptly
Settings for blocked DNS provider returned
Request domain registration
Request for services on the domain
Provision the domain
Provision the NS records
Respond with settings
Set up other DNS records
Respond with settings

Fig. 6: Hardening against DNS Providers suffering outage/downtime

Fig. 6 illustrates hardening against DNS Providers which are experiencing known downtime or suffering outages, per the techniques of this disclosure. In contrast to Fig. 3, the Service Provider can detect, using the described techniques, a blocked DNS Provider and immediately end the flow, thereby preventing existing reliability and security issues from compounding.

Example: An example is provided below that illustrates the solution.

1. Assume that the domain of the customer is “mydomain.com”
2. Assume that the *_domainconnect* record correctly points to the good DNS Provider “MyAwesomeDNSProvider” with the URL “api.myawesomednsprovider.com

3. A Service Provider providing services to “mydomain.com” would query the *_domainconnect* record in point 2 to get “api.myawesomednsprovider.com”
4. The Service Provider would then proceed to make a settings call for mydomain.com by calling the URL *https://api.myawesomednsprovider.com/v2/mydomain.com/settings*
5. This would then contain URLs for integrating with Domain connect like so

```
{
"providerId": "myawesomednsprovider.com",
"providerName": "MyAwesomeDNSProvider",
"providerDisplayName": "MyAwesomeDNSProvider",
"urlSyncUX": "https://domainconnect.myawesomednsprovider.com",
"urlAsyncUX": "https://domainconnect.myawesomednsprovider.com",
"urlAPI": "https://api.domainconnect.myawesomednsprovider.com",
"width": 750,
"height": 750,
"urlControlPanel":
"https://domaincontrolpanel.myawesomednsprovider.com/?domain=mydomain.com",
"nameServers": ["ns01.myawesomednsprovider.com",
"ns02.myawesomednsprovider.com"]
}
```

6. Now the Service Provider, in this proposed scheme would have already contained a list of wildcards for the various values in this settings call to enable them to detect the DNS Provider. For example, consider the below table of values on Service Provider side about DNS Provider:

DNS Provider	**Settings part**	**Settings part pattern**	**Settings part pattern type**
MyAwesomeDNSProvider	providerId	“myawesomednsprovider.com”	SUBSTRING
MyAwesomeDNSProvider	urlSyncUx	“domainconnect.myawesomednsprovider.com”	ENDSWITH

7. Now the Service Provider would match the returned settings call against the known information about providers - since the *providerId* for MyAwesomeDNSProvider is

"*myawesomednsprovider.com*" and the *urlSyncUx* is "*domainconnect.myawesomednsprovider.com" and both patterns match - the DNS Provider is correctly identified as MyAwesomeDNSProvider

8. At this point, since the Service Provider would know which DNS Provider is involved - and any policy or engineering decision about this provider can now be applied, e.g., block it for the duration of the known downtime, outage etc. between the Service Provider and DNS Provider.
9. Another policy decision which can be made is to fallback to settings URLs agreed upon between the Service Provider and the DNS Provider as mentioned earlier in this disclosure.

Solution for problem 3: Hardening against the use of HTTP URLs in settings

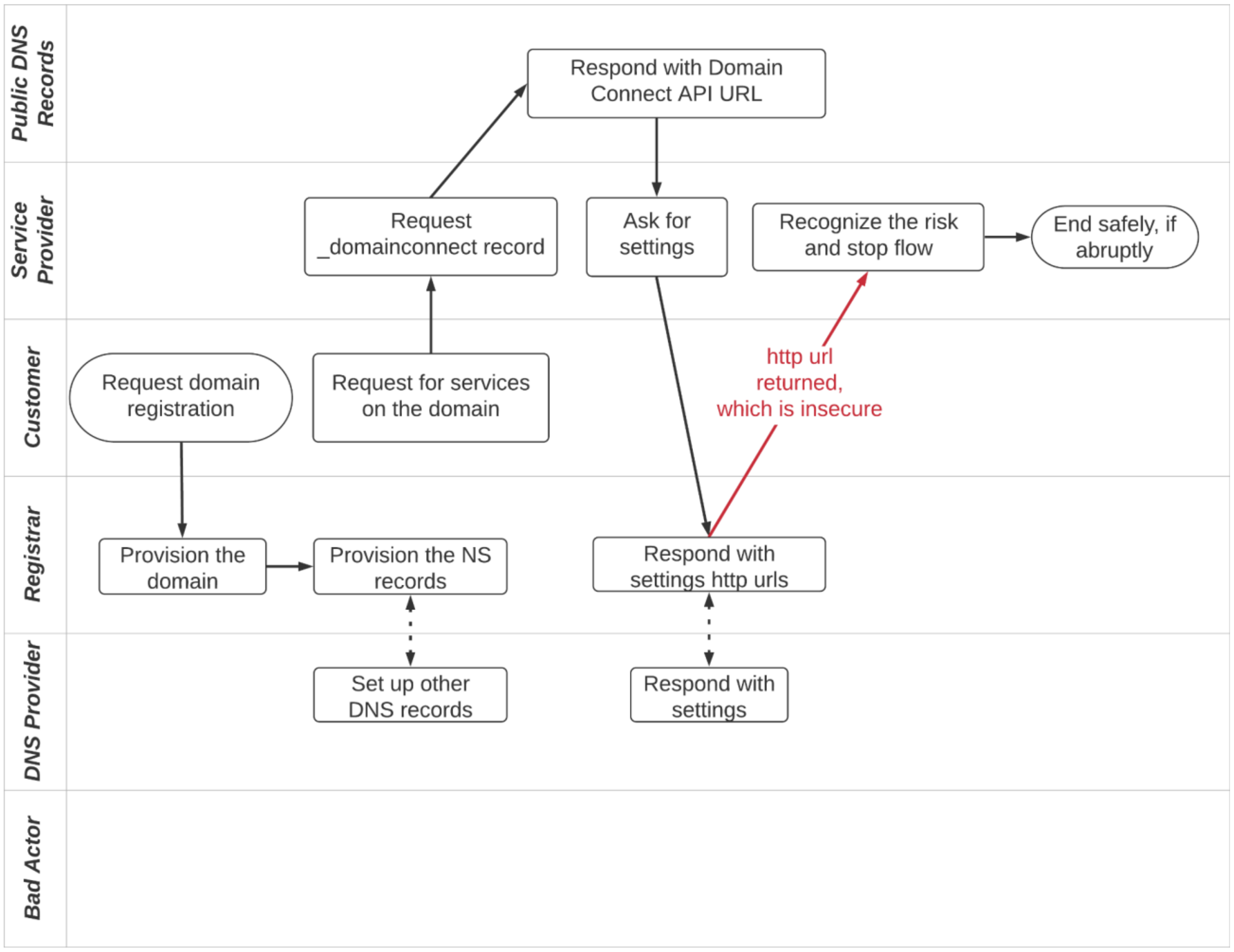
Public DNS Records
Service Provider
Customer
Registrar
DNS Provider
Bad Actor
Respond with Domain Connect API URL
Request _domainconnect record
Ask for settings
Recognize the risk and stop flow
End safely, if abruptly
Request domain registration
Request for services on the domain
http url returned, which is insecure
Provision the domain
Provision the NS records
Respond with settings http urls
Set up other DNS records
Respond with settings

Fig. 7: Hardening against the use of HTTP URLs in settings

Fig. 7 illustrates hardening against the use of HTTP URLs in settings. In contrast to Fig. 4, the Service Provider can detect the use of the HTTP (rather than https) protocol and end the flow, thereby mitigating risk and preventing existing security issues from compounding.

Example: An example is provided below that illustrates the solution.

1. Assume that the domain of the customer is “mydomain.com”
2. Assume that the *_domainconnect* record contains the hostname “api.gooddnsprovider.com” for “mydomain.com”

3. A Service Provider providing services to “mydomain.com” would query the *_domainconnect* record in point 2 to get “api.gooddnsprovider.com”
4. The Service Provider would then proceed to make a settings call for “mydomain.com” by calling the URL *https://api.gooddnsprovider.com/v2/mydomain.com/settings*
5. This response might look like this:

```
{
"providerId": “gooddnsproviderid”,
"providerName": "gooddnsprovider name",
"providerDisplayName": "gooddnsprovider display name",
"urlSyncUX": "http://domainconnect.gooddnsprovider.com",
"urlAsyncUX": "http://domainconnect.gooddnsprovider.com",
"urlAPI": "http://api.domainconnect.gooddnsprovider.com",
"width": 750,
"height": 750,
"urlControlPanel":
"http://domaincontrolpanel.gooddnsprovider.com/?domain=mydomain.com",
"nameServers": ["ns01.gooddnsprovider.com", "ns02.gooddnsprovider.com"]
}
```

6. However, at this very point the Service Provider would check the URLs retrieved in settings call response to ensure that URL follows the *HTTPS* scheme instead of *HTTP*. Since the URL scheme is *HTTP* the Service Provider will not make any subsequent calls- effectively thwarting any man-in-the-middle-attacks before such attacks even start.

CONCLUSION

This disclosure describes an allow-list mechanism that mitigates security and reliability related challenges arising from the current Domain Connect specification. A wildcard (or regular-expression) check is conducted on the initial server URL returned from following the *_domainconnect* TXT record for a given domain. Subsequent wildcard checks also validate fields that are returned in response to the settings call.

REFERENCES

[1] “List of DNS record types” https://en.wikipedia.org/wiki/List_of_DNS_record_types accessed May 26, 2021.

[2] “Domain Connect 2.2,” available online at https://github.com/Domain-Connect/spec/blob/master/Domain%20Connect%20Spec%20Draft.adoc accessed May 18, 2021.