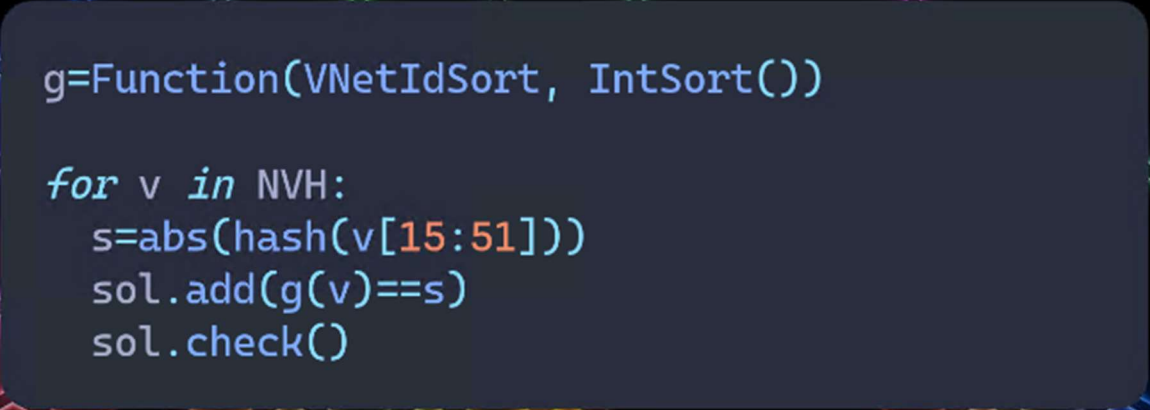


Automated reasoning in Azure and AWS

Version 1.0 (05 May 2022)

Proving the Hub & spoke pattern at scale
With Z3 SMT solver + EUF theory



```
g=Function(VNetIdSort, IntSort())  
  
for v in NVH:  
    s=abs(hash(v[15:51]))  
    sol.add(g(v)==s)  
    sol.check()
```

[linkedin.com/in/parisel](https://www.linkedin.com/in/parisel)

Permalink: <https://github.com/labyrinthinesecurity/automatedReasoning/blob/main/HubSpoke.pdf>

We then coalesce all NHV peers together: for each Vnet ID v_1 in NHV, for each peer v_2 of v_1 , we state the equality relationship $\mathbf{v1 == v2}$.

By virtue of the equality relationship, NHV is now partitioned and all v in NHV are grouped into equivalence classes.

Let's introduce the uninterpreted function $h(\text{VNet ID}) \Rightarrow \text{VNet ID}$

For each Vnet ID v in the NHV partition, if v has at least one peer in H, we state the equality relationship:

$v == h(v)$

By this process, $h(v)$ is added to the equivalence class of v .

$h()$ has the property that if v is a Shallow Vnet, then $v == h(v)$. Note the converse is not always true since $h()$ is uninterpreted.

Also note that all the above can be calculated in $O(n)$, where n is the cardinality of NHV.

We can now prove the SPOKE-TO-HUB property in $O(n)$:

For each VNet ID v in NHV, we add the inequality $v \neq h(v)$ and check if it's satisfiable.

If it is not, we remove the inequality and carry on with the next member of NHV. If it is, then we have found an equivalence class of NHV which doesn't contain a $h(.)$ item. Put it another way, if $v \neq h(v)$ is satisfiable, there exists a group of Non-Hub Vnets represented by v which are not connected to any Shallow VNet or there are no Shallow Vnets in NHV.

At the end of the process, if we haven't managed to satisfy this inequality, then all Non-Hub Vnets are connected to at least one Shallow VNet, either directly or indirectly by transitivity of the equality relationship.

This proves the property at scale.

Before moving on to the NO SPOKE-TO-SPOKE property, we first introduce arithmetization.

ARITHMETIZATION

Let $u()$ be an uninterpreted function with the following signature: *CustomSort* \otimes *Integer* \otimes *Integer* \Rightarrow *Integer*

Definition: for $j > i$ and for any x in CustomSort, $u(x, i, j)$ is arithmetized if $u(x, i, j) == \text{abs}(\text{hash}(\text{str}(x[i:j])))$ in Python.

Intuitively, **$u()$ is an oracle** that assigns a unique, deterministic integer to an uninterpreted symbol.

NO SPOKE-TO-SPOKE property

Let's introduce an arithmetized oracle $o()$ with signature: *SubscriptionId* \Rightarrow *Integer*

To that end, we set $i=0$ and $j=-1$ so we have $o(\text{subscriptionId}, 0, -1) = \text{abs}(\text{hash}(\text{str}(\text{subscriptionId}[0:-1])))$

Since $\text{string}[0:-1] == \text{string}$, we may drop i and j , and $o(.)$ simply returns the hash of the subscription ID.

For each subscription ID subId in NHV Vnets, we add the following oracular constraint to the Z3 model:

$\text{subId} == \text{abs}(\text{hash}(\text{subId}))$ [equation 0]

We now introduce a second oracle, $g()$ of type $VNetIdSort \otimes Integer \otimes Integer \Rightarrow Integer$

In the case of Azure, we set $i = 15$ and $j = 51$ so that for any VNetId v , $g(v, i, j)$ is the arithmetization of the subscription ID of v (shown in red below between characters 15 and 51 included):

`/subscriptions/00000000-aaaa-bbbb-cccc-
ffffffffffff/resourceGroups/RG/providers/Microsoft.Network/virtualNetworks/v`

This is not so straightforward in AWS since VPC IDs do not contain the account number, but an equivalent key can be constructed by sticking the account ID and the VPN ID for instance.

By nature of Python's `hash()` function, it's easy to see that the following property is true with overwhelming probability:

For any two VnetIdSort symbols $g(x, 15, 51) \neq g(y, 15, 51) \Rightarrow x \neq y$ [equation 1]

All remains to be done is apply $g(.)$ sequentially to each VNet Id v in NHV's partition and see what happens:

$g(v, 15, 51) == \text{abs}(\text{hash}(\text{str}(v[15:51])))$ [equation 2]

If [equation 2] is not satisfiable for a given v , it means that $g(v, 15, 51)$ has already been assigned another value, so the NO SPOKE-TO-SPOKE PROPERTY is disproved.

Why?

- For any v_prime in the same class as v , we have $v == v_prime$ by construction of the NHV partition
- During iteration over [equation 0], some v_prime in the same equivalence class as v was already resolved as $\text{abs}(\text{hash}(\text{str}(v_prime[15:51])))$ which is different from $\text{abs}(\text{hash}(\text{str}(v[15:51])))$ if v and v_prime belong to distinct subscriptions
- If v and v_prime belong to distinct subscriptions while belonging to the same equivalence class, we get $v == v_prime$ and $g(v, 15, 51) \neq g(v_prime, 15, 51)$ at the same time, which is not possible by [equation 1]

If [equation 2] is satisfiable for all VNet ID in NHV, the NO SPOKE-TO-SPOKE PROPERTY is proven.

Once again, all operations can be carried out in $O(n)$, ensuring scalability of the proof.

APPENDIX: Z3 pseudo code

Spoke to hub property

```
from z3 import *
sol=Solver()
v1,v2=Consts('v1 v2',VNetIdSort)
for v1 in peers:
    globals()[v1]=None
    v1=Const(v1,VNetIdSort)
    for v2 in peers[v1]:
        globals()[v2]=None
        v2=Const(v2,VNetIdSort)
        sol.add(v1==v2)

h=Function(VNetIdSort, VNetIdSort)
for v in NHV:
    sol.push()
    sol.add(v!=h(v))
    sol.check()
    sol.pop()
```

No spoke-to-spoke property (note that indexes i==15 and j==51 have been replaced by constants)

```
g=Function(VNetIdSort, IntSort())
for s in NHV's subscription IDs:
    subIdHash=abs(hash(str(s)))
    sol.add(s==subIdHash)

for v in NVH:
    s=abs(hash(v[15:51]))
    sol.add(g(v)==s)
    sol.check()
```