ADCS: A Love Story

Not really

Rohan Vazarkar

@CptJesus
Staff Software Developer
BloodHound Enterprise Team

The Problem

“When is BloodHound going to support ADCS?”

“ADCS when?”

“Is BloodHound going to get support for ADCS?”

“Can we get an estimate on when ADCS will get implemented???”

Initial Refinement

- Jonas Knudsen (@jonasbk) and Andy Robbins (@_wald0) spent a lot of time before engineering work started parsing ADCS research into what the graph should look like
- Determined which escalations would fit in our graph, as well as the intermediary steps necessary for every single escalation
- Determined what data would need to be collected from AD to make sure we had everything accurate

Initial Estimates

- Sitting in the Seattle office with Jonas, talking about what's necessary for ADCS, thought it wouldn't be so bad
- Filled with hopeful optimism
- Started work, made some progress, estimated that we would be done Q4 of last year
- Started talking about ADCS publicly

Crushing Defeat

- Turns out, we're (honestly, everyone is) really bad at estimating
- ADCS ended up being about 50 times more complex than we anticipated
- Some of our fundamental assumptions ended up being wrong
- Our traversal engine didn't even support some of the things necessary to do ADCS post processing
- Initial prototypes of ADCS were blowing out RAM on fairly small test environments
- Utter complete sadness

ADCS Complexity

- ADCS post processing is the most complex logic we've ever put into BloodHound
- Previous "complex" edges like DCSync/SyncLapsPassword were a combination of 2 privileges
- ADCS edges are combinations of 3-5 permission sets

```
MATCH p1 =
(n:Base)-[:Enroll|GenericAll|AllExtendedRights|MemberOf*1..]->(ct:CertTemplate)-[:PublishedTo]-(ca:EnterpriseCA)-[:IssuedSignedBy*1..]-(rootca:RootCA)-[:RootCAFor]-
>(d:Domain)

MATCH p2 = (n)-[:Enroll|GenericAll|AllExtendedRights|MemberOf*1..]->(ca)-[:TrustedForNTAuth]->(nt:NTAuthStore)-[:NTAuthStoreFor]-(d)
```

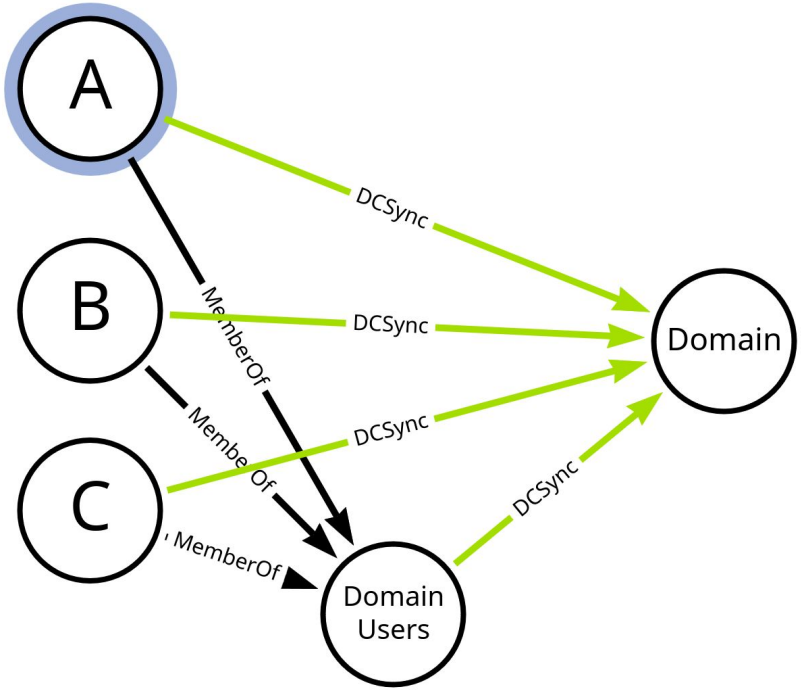
The Easy Way

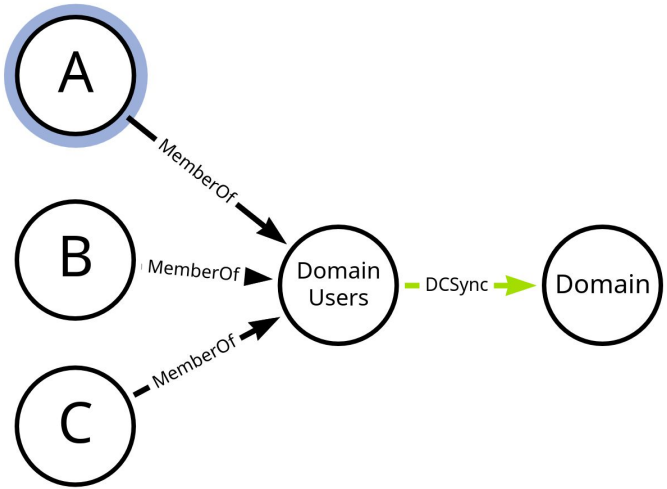
- Find all the first degree controllers of node A and node B
- Expand all group members recursively
- Find all the intersections between the two
- Make an edge from each principal found this way as an attack path

The hard, more right way

- Use the compressed bitmap to store all our group membership expansions to reduce expensive traversal time
- Implement shortcutting logic to opportunistically create edges at the highest possible level, minimizing the number of created edges and taking advantage of graph semantics
  - If the Domain Users group has the permission, all the members of the group have the permission via the MemberOf edge in the graph
  - Cost - some accuracy is lost as some ancillary, valid attack paths can be obscured by the shortcutting

Shortcutting - These graphs mean the same thing!





Edge Composition

- Post processing is hard
- Performantly evaluating the entire dataset to make sure that we catch edge cases takes effort, but we’re working with no start/end point so we can just batch up logic
- Edge composition is even harder as its restricted to a specific set of nodes instead of just being able to use large sets of nodes

```
MATCH (u:User {objectid:'S-1-5-21-2057499049-1289676208-1959431660-238209'})-[:ADCSESC1]->(d:Domain {objectid:'S-1-5-21-1621856376-872934182-3936853371'})
MATCH (c:Container)-[:Contains]->(rca:RootCA)
WHERE c.name CONTAINS "CERTIFICATION AUTHORITIES" AND c.domain = d.domain
MATCH (ca:EnterpriseCA)-[:IssuedSignedBy|EnterpriseCAFor*1..]->(rca)
WHERE (ca)-[:TrustedForNTAuth]->(:NTAuthStore)
MATCH (ct:CertTemplate)-[:PublishedTo]->(ca)
WHERE (ct.requiresmanagerapproval = false
AND ct.schemaversion > 1
AND ct.authorizedsignatures = 0
AND ct.authenticationenabled = true
AND ct.enrolleesuppliessubject = true)
OR (ct.requiresmanagerapproval = false
AND ct.schemaversion = 1
AND ct.authenticationenabled = true
AND ct.enrolleesuppliessubject = true)
OPTIONAL MATCH p1 = (u)-[:GenericAll|Enroll|AllExtendedRights]->(ct)-[:PublishedTo]->(ca)
OPTIONAL MATCH p2 = (u)-[:MemberOf*1..]->(:Group)-[:GenericAll|Enroll|AllExtendedRights]->(ct)-[:PublishedTo]->(ca)
OPTIONAL MATCH p3 = (u)-[:Enroll]->(ca)-[:IssuedSignedBy|EnterpriseCAFor|RootCAFor*1..]->(d)
OPTIONAL MATCH p4 = (u)-[:MemberOf*1..]->(:Group)-[:Enroll]->(ca)-[:IssuedSignedBy|EnterpriseCAFor|RootCAFor*1..]->(d)
OPTIONAL MATCH p5 = (ca)-[:TrustedForNTAuth]->(:NTAuthStore)-[:NTAuthStoreFor]->(d)
RETURN p1,p2,p3,p4,p5
```

Fix it!

- Our traversal engine for cypher was missing several pieces necessary for these
- Spent 2 weeks just rewriting our engine to allow us to begin doing edge composition - pattern continuations allow us to write GO code that mimics the cypher previously shown in a very logical way

```
func adcsESC13Path1Pattern() traversal.PatternContinuation {
    return traversal.NewPattern().
        OutboundWithDepth(
            min: 0, max: 0,
            query.And(
                query.Kind(query.Relationship(), ad.MemberOf),
                query.Kind(query.End(), ad.Group),
            ),
        ).
        OutboundWithDepth(
            min: 1, max: 1,
            query.And(
                query.KindIn(query.Relationship(), ad.Enroll, ad.GenericAll, ad.AllExtendedRights),
                query.KindIn(query.End(), ad.CertTemplate),
                query.And(
                    query.Equals(query.EndProperty(ad.AuthenticationEnabled.String()), value: true),
                    query.Or(
                        query.Equals(query.EndProperty(ad.SchemaVersion.String()), value: 1),
                        query.And(
                            query.GreaterThan(query.EndProperty(ad.SchemaVersion.String()), value: 1),
                            query.Equals(query.EndProperty(ad.AuthorizedSignatures.String()), value: 0),
                        ),
                    ),
                    query.Equals(query.EndProperty(ad.RequiresManagerApproval.String()), value: false),
                ),
            ),
        ).OutboundWithDepth(
        min: 1, max: 1,
        query.And(
            query.KindIn(query.Relationship(), ad.PublishedTo),
            query.Kind(query.End(), ad.EnterpriseCA),
        )).
        Outbound(query.And(
            query.KindIn(query.Relationship(), ad.IssuedSignedBy, ad.EnterpriseCAFor),
            query.Kind(query.End(), ad.RootCA),
        )).
        Outbound(query.And(
            query.KindIn(query.Relationship(), ad.RootCAFor),
            query.KindIn(query.End(), ad.Domain),
        ))
}
```

Did it work?

- Wrote the first edge composition POC - immediately killed our test instance by blowing out memory
- Rewrote our edge composition completely using incredibly gross optimizations - finally it worked!
  - Opportunistic pruning of target nodes
  - Starting from the middle of a query and working backwards to reduce total path space

The problem?

- We’re now late Q4 of 2023.
- We’re just wrapping up ESC1. We need to add ESC3,4,6,9,10,13
- Jonas and Andy are still finding more escalations
- Clearly our timelines were correct
- Finally just accepted that we couldn’t deliver Q4, doubled down on the work and got it all done in Q1 of 2024 with extra engineering effort

Take-aways

- Complex problems are extremely difficult to estimate timelines for
- Some problems are vastly more complex than they seem on the surface
- Engineers think they can solve everything quickly and perfectly (especially me)
- I’m just rambling at this point