



Will

[Follow](#)

Co-founder of Empire/BloodHound/Veil-Framework | PowerSploit developer | Microsoft PowerShell MVP | Security at the misfortune of others | <http://specterops.io>

Oct 30, 2017 · 36 min read

A Guide to Attacking Domain Trusts

It's been a while (nearly 2 years) since I wrote a post purely on Active Directory domain trusts. After diving into group scoping, I realized a few subtle misconceptions I previously had concerning trusts and group memberships. That, combined with the changes made to PowerView last year, convinced me to publish an up-to-date guide on enumerating and attacking domain trusts. This will likely be the last post focusing on domain trusts I publish for a while, and at over 8000 words, it's not exactly a light read (not that anyone reads long posts ;)) In general, I don't just blog my operational notes—I try to write posts that function as complete guides for members of my team, with the hope that the information may be of use to others (whether from an offensive or defensive perspective.)

I want this to be as complete as possible, so I'll cover every aspect of trusts as we currently understand them. Just as with my previous posts, I want to

encapsulate my knowledge of the topic as best I can *at this point in time*. Emphasis on “this point in time.” Our knowledge and tradecraft are always evolving, and trusts are no different. I had a number of fuzzy misconceptions regarding domain trusts when I started writing about them. I was never a sysadmin or AD architect—I’ve learned my knowledge piecemeal, which (hopefully) explains the gaps that have surfaced in my past posts, and the ones that I’m sure will continue to arise.

So I am going to start fresh, in case you are not familiar with the previous posts I pushed out about trusts. As such, a few parts of this post will recycle certain elements and wording from previous work, integrated with updated knowledge and PowerView syntax. And as this is a bit of a tome of an article, I’m sure there are mistakes somewhere and things that I’ve missed, so when you find them let me know and I’ll update appropriately!

PowerView’s most up-to-date version will always be on the dev branch of PowerSploit.

WTF Are Domain Trusts?

At a high level, a **domain trust** establishes the ability for users in one domain to authenticate to resources or act as a security principal in another domain. Microsoft has a lot of information out there about domain trusts, as well as “Security Considerations for Trusts”, and it can sometimes get a bit confusing. As Microsoft describes, “*Most organizations that have more than one domain have a legitimate need for users to access shared resources located in a different domain*“, and trusts allow organizations with multiple domains to grant users in separate domains access to shared resources. **Domain forests** are collections of domain containers that trust each other. Forests themselves may also have trusts between them. Microsoft has excellent post about how domain and forest trusts work. If you’re not familiar with this topic, I recommend that you check it out.

Essentially, all a trust does is link up the authentication systems of two domains and allows authentication traffic to flow between them through a system of referrals. If a user requests access to a service principal name (SPN) of a resource that resides outside of the domain they’re current in, their domain controller will return a special referral ticket that points to the key distribution center (KDC, in the Windows case the domain controller) of the foreign domain.

The user's ticket-granting-ticket (TGT) is included in this TGS-REP (ticket-granting service reply) referral ticket, and this ticket encrypted/signed with the inter-realm trust key that the domains previously exchanged, instead of the first domain's krbtgt account. This ticket is usually referred to as an "inter-realm ticket-granting-ticket/TGT." The foreign domain then verifies/decrypts the TGT included in the referral by decrypting it with the previously negotiated inter-realm trust key, and goes about the rest of the normal Kerberos process.

Sean Metcalf has a great breakdown of this process in his "It's All About Trust" post, and he describes the process as, *"Once there is a trust between two domains ... the ticket-granting service of each domain ("realm" in Kerberos speak) is registered as a security principal with the other domain's Kerberos service (KDC). This enables the ticket-granting service in each domain to treat the one in the other domain as just another service providing cross-domain service access for resources in the other domain."*

So basically, when the foreign domain decrypts the referral ticket with the negotiated trust key, it sees the user's TGT and says *"OK, the other domain already authenticated this user and said this is who they say they are/these are*

the groups the user is in, so I'll trust this information as accurate because I trust the domain that issued the referral."

Here's a picture to visualize the Kerberos process across trust boundaries:



The purpose of establishing a trust is to allow users from one domain to access resources (like the local Administrators group on a server), to be nested in groups, or to otherwise be used as security principals in another domain (e.g. for AD object ACLs). One exception to this is intra-forest trusts (domain trusts that exist within the same Active Directory forest)- any domain created within a forest retains an implicit two-way, transitive trust relationship with every other domain in the forest. This has numerous implications which will be covered later in this post.

But before that, we have to cover a few more characteristics of trusts.

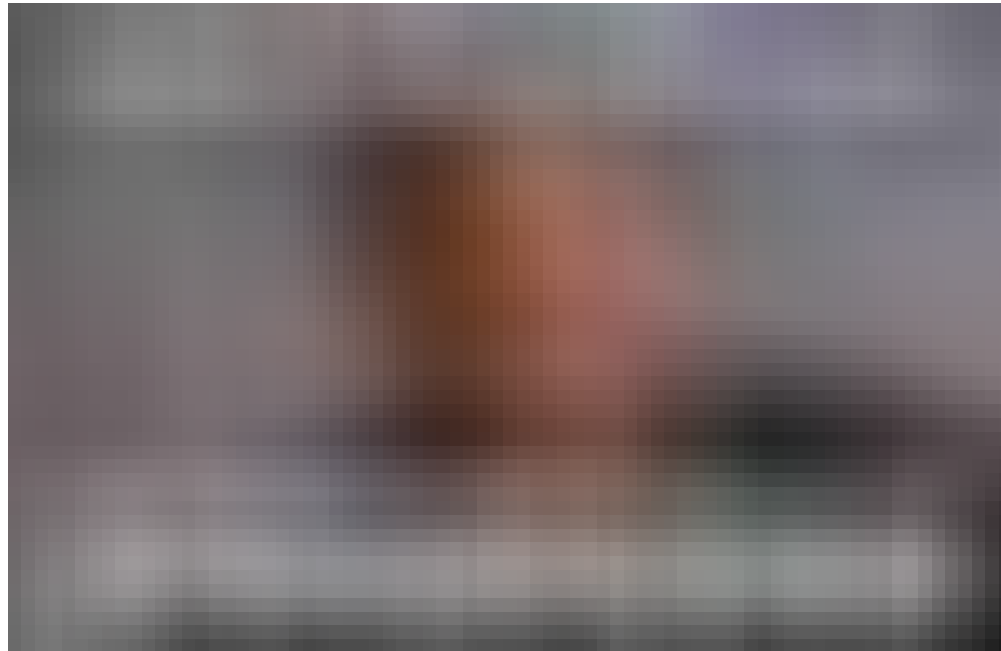
There are several types of trusts, some of which have various offensive implications, covered in a bit:

- **Parent/Child**—part of the same forest—a child domain retains an implicit two-way transitive trust with its parent. This is probably the most common type of trust that you'll encounter.
- **Cross-link**—aka a “shortcut trust” between child domains to improve referral times. Normally referrals in a complex forest have to filter up to the forest root and then back down to the target domain, so for a

geographically spread out scenario, cross-links can make sense to cut down on authentication times.

- **External**—an implicitly non-transitive trust created between disparate domains. *“External trusts provide access to resources in a domain outside of the forest that is not already joined by a forest trust.”* External trusts enforce SID filtering, a security protection covered later in this post.
- **Tree-root**—an implicit two-way transitive trust between the forest root domain and the new tree root you’re adding. I haven’t encountered tree-root trusts too often, but from the [Microsoft documentation](#), they’re created when you when you create a new domain tree in a forest. These are intra-forest trusts, and they preserve two-way transitivity while allowing the tree to have a separate domain name (instead of child.parent.com).
- **Forest**—a transitive trust between one forest root domain and another forest root domain. Forest trusts also enforce SID filtering.
- **MIT**—a trust with a non-Windows [RFC4120-compliant](#) Kerberos domain. I hope to dive more into MIT trusts in the future.

Transitivity, huh? Another aspect of domain trusts is that they are transitive or non-transitive. To quote the MSDN documentation on transitivity: “A transitive trust extends trust relationships to other domains; a nontransitive trust does not extend trust relationships to other domains.” This means that transitive trusts can be chained, so users can potentially access resources in multiple domains. Meaning, if domain A trusts B, and B trusts C, then A implicitly trusts C. Under the hood, if a specific trust relationship is transitive, then the *trusting* domain can repack a user’s TGT into additional referral tickets and forward them onto domains *that* domain trusts.



Also, trusts can be one-way or two-way. A bidirectional (two-way) trust is actually just two one-way trusts. A one-way trust means users and computers in a *trusted domain* can potentially access resources in another *trusting domain*. A one-way trust is in one direction only, hence the name. Users and computers in the *trusting* domain can not access resources in the *trusted* domain. Microsoft has a nice diagram to visualize this:



[https://technet.microsoft.com/en-us/library/cc759554\(v=ws.10\).aspx](https://technet.microsoft.com/en-us/library/cc759554(v=ws.10).aspx)

This was something that messed with my head when I started—from an offensive perspective, what we care about is the *direction of access*, not the *direction of the trust*. With a one-way trust where **A -trusts-> B**, if the trust is enumerated from A, the trust is marked as *outbound*, while if the same trust is enumerated from B the trust is marked as *inbound*, while the potential access is from **B** to **A**. This will make more sense in the **Foreign Relationship Enumeration** section.

Why Care?

But really, why care?

Domain trusts often introduce unintended access paths between environments. In many organizations, trusts were implemented years (sometimes 10+) ago without major considerations given to security. Some corporate entities that are focused on acquisitions often just “plug in” a new company’s Active Directory network either as a child domain or external trust, without fully considering the security implications.

Because historically there have not been many toolsets that allow you to easily map, enumerate, and visualize the risk associated with misconfigured trusts,

many domain architects are unaware of the unintentional risk exposed by their Active Directory trust architectures. This links back to the idea of “misconfiguration debt” that [@wald0](#), [@cptjesus](#), and I [spoke about at Derbycon this year](#). Because of this, various red teams (and probably APTz, I’m assuming) have been abusing Active Directory trusts for years with great success.

A common scenario is compromising a development or subsidiary domain and leveraging that access to pivot into the secure root/enclave. This also introduces opportunities for persistence- why leave code running in a secured environment, when you can have implants running in the less-secured (but trusted) domain that can then be used to re-compromise your target at will?

An intra-forest trust (parent/child or tree-root) introduces an awesome attack vector that’s described in **The Trustpocalypse** later in this post.

External/inter-forest trusts *do not guarantee* any kind of privileged access, but at a minimum, a trust means that you can query any normal Active Directory information from a domain that trusts you (yes, this means in some cases you can Kerberoast across trusts, more on this at the end of post.) After all, Active

Directory is meant as a queryable database of information, and trusts don't change that!

A Trust Attack Strategy

Before we get into the technical details of how to enumerate and abuse trusts, I wanted to go over the high level strategy I use when auditing trust relationships. When I talk about a “trust attack strategy” what I mean is a way to laterally move from the domain in which your access currently resides into another domain you're targeting.

(1) The first step is to enumerate all trusts your current domain has, along with any trusts *those* domains have, and so on. Basically, you want to produce a mapping of all the domains you can reach from your current context through the linking of trust referrals. This will allow you to determine the domains you need to hop through to get to your target and what techniques you can execute to (possibly) achieve this. Any domains in the mapped “mesh” that are in the same forest (e.g. parent->child relationships) are of particular interest due to the SIDhistory-trust-hopping technique developed by Sean Metcalf and Benjamin Delpy, also covered in the **The Trustpocalypse** section.

(2) The next step is to enumerate any users/groups/computers (security principals) in one domain that either (1) have access to resources in another domain (i.e. membership in local administrator groups, or DACL ACE entries), or (2) are in groups or (if a group) have users from another domain. The point here is to find relationships that cross the mapped trust boundaries in some way, and therefore might provide a type of “access bridge” from one domain to another in the mesh. While a cross-domain nested relationship is not guaranteed to facilitate access, trusts are normally implemented for a reason, meaning more often than not some type of cross-domain user/group/resource “nesting” probably exists, and in many organizations these relationships are misconfigured. Another subnote- as mentioned, Kerberoasting across trusts **may** be another vector to hop a trust boundary. Check out the **Another Sidenote: Kerberoasting Across Domain Trusts** section for more information.

(3) Now that you have mapped out the trust mesh, types, and cross-domain nested relationships, you have a map of what accounts you need to compromise to pivot from your current domain into your target. By performing targeted account compromise, and utilizing SID-history-hopping

for domain trusts within a forest, we have been able to pivot through up to 7+ domains in the field to reach our objective.

At a minimum, remember that if a domain trusts you, i.e. if the trust is bidirectional or if one-way and inbound, then you can query any Active Directory information from the *trusting* domain. And remember that all parent->child (intra-forest domain trusts) retain an implicit two way transitive trust with each other. Also, due to how child domains are added, the “Enterprise Admins” group is automatically added to Administrators domain local group in each domain in the forest. This means that trust “flows down” from the forest root, making it our objective to move from child to forest root at any appropriate step in the attack chain.

OK, How Do I Enumerate Trusts?

OK Will, you’ve piqued my interest. How do I go about figuring out what trust relationships exist in my environment?

As far as I know, there are three main methods to enumerate trusts: Win32 API calls, various .NET methods, and LDAP. Each one (frustratingly) returns a differing set of information, and each one has different execution methods. I’ll

cover the old school ways and the new, from built-in (and external) binaries, to .NET, to Win32 API calls, to PowerShell/PowerView and BloodHound.

The sample trust architecture I'll be using for this post is:



This image was generated with the new TrustVisualizer output (described in the **Visualizing Domain Trusts** section). With this new output, **green** edges mean “within forest”, **red** means external, and **blue** means inter-forest trust relationships. As with @sixdub's DomainTrustExplorer the edge directions for one-way trust mean *direction of access*, not *direction of trust*.

.NET Methods

.NET provides us with some nice method wrappers that can enumerate a good chunk of domain and forest trust information. This was the first method that PowerView implemented, before branching into Win32 API and LDAP methods.

The [System.DirectoryServices.ActiveDirectory.Domain] namespace has a GetCurrentDomain() static method that returns a System.DirectoryServices.ActiveDirectory.Domain class instance. This class implements the GetAllTrustRelationships() method which nicely, “*Retrieves all of the trust relationships for this domain.*” One advantage of this method is its simplicity—the information is laid out in a fashion that is easy to read and understand. One disadvantage is that it doesn’t contain some of the additional information that other enumeration methods produce.

```
([System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()).GetAllTrustRelationships()
```

This used to be PowerView’s default **Get-DomainTrust** enumeration method. I recently changed the default method to be LDAP, as this .NET method does

not return forest trusts by default, while LDAP enumeration does. So in order to execute this method, you now need to run **Get-DomainTrust -NET**.

Here's how it looks for my sample domain setup, running the enumeration from **sub.dev.testlab.local**:



Forest trusts are functionally different than domain trusts. So if you want to enumerate any current forest->forest trusts, you need to call on [System.DirectoryServices.ActiveDirectory.Forest] instead. Resulting forest objects also have their own GetAllTrustRelationships() method which will return any current forest trusts:

```
([System.DirectoryServices.ActiveDirectory.Forest]::GetCurrentForest()).GetAllTrustRelationships()
```

This is implemented as the default enumeration methods for PowerView's **Get-ForestTrust** function. Here's how it looks for my sample domain setup, again from **sub.dev.testlab.local**:



Win32API

You can also enumerate domain trusts through the DsEnumerateDomainTrusts() Win32 API call which returns a DS_DOMAIN_TRUSTS structure. While the information is a bit more complex than the .NET methods, it returns the SID and GUID of the target domain, as well as some useful flags and attributes. The flags are documented here and will tell you the trust direction, whether the trust is within the same forest, etc. The attributes are documented here under the **TrustAttributes**

specification, and include things like `WITHIN_FOREST`, `NON_TRANSITIVE`, `FILTER_SIDS`, and more. `FILTER_SIDS` is the equivalent of `QUARANTINED_DOMAIN` if you ever see that nomenclature.

You can invoke this method with **Get-DomainTrust -API** (same `sub.dev.testlab.local` origination domain):





Of note, this appears to be what **nltest.exe** uses with its **/trusted_domains** flag:



This is also the method that BloodHound uses to enumerate domain trusts. You can execute this with the new SharpHound.ps1 ingestor by using the **Invoke-BloodHound -CollectionMethod trusts** syntax. Note that this can also be combined with **-Domain <foreign.domain.fqdn>** for foreign trust enumeration as well.

LDAP

Domain trusts are stored in Active Directory as “trusted domain objects” with an objectClass of **trustedDomain**. This means you can use whatever LDAP querying method you would like to find out information about any domain trusts that are present by using the LDAP filter **(objectClass=trustedDomain)**.


For example, here’s dsquery (only available on Windows servers):

```
dsquery * -filter “(objectClass=trustedDomain)” -attr *
```



The equivalent syntax with Joeware's Adfind is `.\adfind.exe -f
objectclass=trusteddomain`.

And finally PowerView, which again now uses this LDAP as the default enumeration method for **Get-DomainTrust**:



Since this LDAP method is now the default for PowerView's **Get-DomainTrust**, I'm going to break down some of the result properties that might be a bit confusing.

TrustType:

- **DOWNLEVEL** (0x00000001)—a trusted Windows domain that IS NOT running Active Directory. This is output as **WINDOWS_NON_ACTIVE_DIRECTORY** in PowerView for those not as familiar with the terminology.
- **UPLEVEL** (0x00000002)—a trusted Windows domain that IS running Active Directory. This is output as **WINDOWS_ACTIVE_DIRECTORY** in PowerView for those not as familiar with the terminology.
- **MIT** (0x00000003)—a trusted domain that is running a non-Windows (*nix), RFC4120-compliant Kerberos distribution. This is labeled as MIT due to, well, MIT publishing RFC4120.

TrustAttributes:

- **NON_TRANSITIVE** (0x00000001)—the trust cannot be used transitively. That is, if DomainA trusts DomainB and DomainB trusts DomainC, then DomainA does not automatically trust DomainC. Also, if a trust is non-transitive, then you will not be able to query any Active Directory information from trusts up the chain from the non-transitive point. External trusts are implicitly non-transitive.
- **UPLEVEL_ONLY** (0x00000002)—only Windows 2000 operating system and newer clients can use the trust.
- **QUARANTINED_DOMAIN** (0x00000004)—SID filtering is enabled (more on this later). Output as **FILTER_SIDS** with PowerView for simplicity.
- **FOREST_TRANSITIVE** (0x00000008)—cross-forest trust between the root of two domain forests running at least domain functional level 2003 or above.
- **CROSS_ORGANIZATION** (0x00000010)—the trust is to a domain or forest that is not part of the organization, which adds the

OTHER_ORGANIZATION SID. This is a bit of a weird one. I don't remember encountering this flag in the field, but according to [this post](#) it means that the selective authentication security protection is enabled. For more information, check out [this MSDN doc](#).

- **WITHIN_FOREST** (0x00000020)—the trusted domain is within the same forest, meaning a parent->child or cross-link relationship
- **TREAT_AS_EXTERNAL** (0x00000040)—the trust is to be treated as external for trust boundary purposes. According to [the documentation](#), *“If this bit is set, then a cross-forest trust to a domain is to be treated as an external trust for the purposes of SID Filtering. **Cross-forest trusts are more stringently filtered than external trusts**. This attribute relaxes those cross-forest trusts to be equivalent to external trusts.”* This sounds enticing, and I'm not 100% sure on the security implications of this statement ^_('▽)_/ ^ but I will update this post if anything new surfaces.
- **USES_RC4_ENCRYPTION** (0x00000080)—if the TrustType is MIT, specifies that the trust that supports RC4 keys.
- **USES_AES_KEYS** (0x00000100)—not listed in the linked Microsoft documentation, but according to [some documentation](#) I've been able to

find online, it specifies that AES keys are used to encrypt KRB TGTs.

- **CROSS_ORGANIZATION_NO_TGT_DELEGATION** (0x00000200)—*“If this bit is set, tickets granted under this trust MUST NOT be trusted for delegation.”* This is described more in [\[MS-KILE\] 3.3.5.7.5](#) (Cross-Domain Trust and Referrals.)
- **PIM_TRUST** (0x00000400)—*“If this bit and the TATE (treat as external) bit are set, then a cross-forest trust to a domain is to be treated as Privileged Identity Management trust for the purposes of SID Filtering.”* According to [\[MS-PAC\] 4.1.2.2](#) (SID Filtering and Claims Transformation), *“A domain can be externally managed by a domain that is outside the forest. The trusting domain allows SIDs that are local to its forest to come over a PrivilegedIdentityManagement trust.”* While I have not seen this in the field, and it’s only supported by domain functional level 2012R2 and above, it also warrants further investigation :)

All of these methods can also be executed against a domain that currently trusts you. Meaning, if your current domain has a bidirectional trust with FOREIGN domain, or if the trust is one-way and inbound (meaning said domain trusts you and therefore you have some kind of access), you can

execute these methods against said domain to find the trusts for THAT domain. If you want to do this with PowerView, just supply the **-Domain** `<domain.fqdn>` parameter, described in more detail in the next section.

Data Enumeration Across Trusts With PowerView

Last year I described my ground-up rewrite of PowerView. One of the changes mentioned was that now, any **Get-Domain*** function uses LDAP enumeration, meaning that we can pull said information from a domain that trusts us. This is done with the **-Domain** `<domain.fqdn>` parameter:





So what's actually happening under the hood?

For a long time, I thought that this would “reflect” LDAP queries through a domain controller in your current domain and onto domain controllers in the trusting domain. This would have been an awesome way to get around network boundaries, but sadly I was mistaken. What actually happens is that a referral is returned by the domain controller you are currently communicating with, which instructs your searching method to then bind to the foreign domain (i.e. the primary domain controller/PDC for that domain). If there is a trust with the foreign domain, an inter-realm TGT will be returned that can be used when communicating to the foreign domain.

This means that if there is network segmentation between the computer you're currently querying from, and the PDC for the trusting domain, you won't be able to retrieve any results >_<

From the Kerberos side, “under the hood”, this means that a series of inter-realm referral tickets are automatically issued that allow our user to

eventually request an LDAP service ticket from the target domain. If we use our sample domain architecture, and currently reside in **sub.dev.testlab.local** while querying **prod.contoso.local**, here's how the klist output looks:



You can see the inter-realm tickets filtering up the trust chain to **testlab.local**, and eventually to **contoso.local** and eventually to **prod.contoso.local**.

Mapping Domain Trusts

There are few ways I know of to map the “mesh” of one or more trusts that exist in your environment. The first is through the global catalog. I talked about this a bit in my [“A Pentester’s Guide to Group Scoping”](#) post, but I’ll reiterate some of that information here.

The global catalog is a partial copy of all objects in an Active Directory forest, meaning that some object properties (but not all) are contained within it. This data is replicated among all domain controllers marked as global catalogs for the forest. Trusted domain objects are replicated in the global catalog, so we can enumerate every single internal and external trust that all domains in our current forest have extremely quickly, and only with traffic to our current PDC by running **Get-DomainTrust -SearchBase “GC://\$((\$ENV:USERDNSDOMAIN))”** through PowerView. Here’s how that looks running that function from **sub.dev.testlab.local** domain:





This is a lot more results than just **Get-DomainTrust** !

The second method is slower, but will provide even more results. Since we can enumerate any trusts that our current domain context has, and by way of referrals through LDAP, we can query any (**objectClass=trustedDomain**)

objects from domains that currently trust our domain, then we can keep issuing these queries for any results and “crawl” any reachable domains. Any domains marked as non-transitive can mess these results up, but we can still get a good number of results.

The PowerView function to do this is **Get-DomainTrustMapping** (formerly Invoke-MapDomainTrust). These results can be exported to a CSV by piping **Get-DomainTrustMapping** to **| Export-CSV -NoTypeInfoInformation trusts.csv**.

The last way is through BloodHound/SharpHound. Again, you can execute this with the new SharpHound.ps1 ingestor by using the **Invoke-BloodHound -CollectionMethod trusts** syntax, and this can be combined with **-Domain <foreign.domain.fqdn>** for foreign trust enumeration.

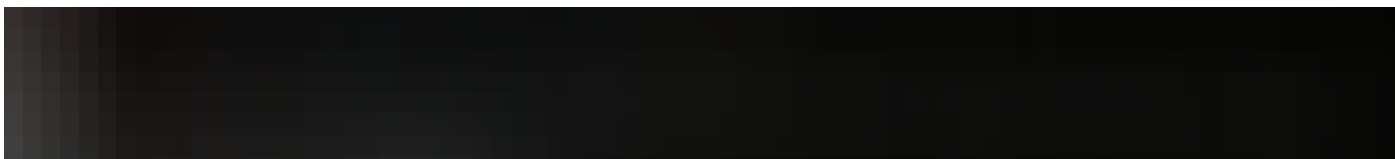
A key thing to remember is that the exact trust mapping you’ll get will depend on the domain you’re currently in. Since the trust between **external.local** and **sub.dev.testlab.local** domain is a one-way *non-transitive* external trust, if you’re querying from **external.local** you won’t be able to see the trusts that **contoso.local** has, again because **sub.dev.testlab.local** won’t repackage your

TGT into an inter-realm TGT that can be forwarded onto any other domain. Also, if you're trying to enumerate the trusts on a foreign domain, you need to be able to bind to a domain controller (usually the PDC/primary domain controller) in the foreign domain you're querying. So, even if there is a transitive trust that would allow you to query the information, if network segmentation prevents you from talking to the target foreign domain, you're out of luck.

Visualizing Domain Trusts

Data is one thing, visualizations are another. A few years ago, one of my former workmates, Justin Warner, saw all this raw data and build a tool called DomainTrustExplorer that could perform some nodal analysis and visualization with PowerView's mapped trust data.

As the default trust output has changed, and with BloodHound taking care of nodal analysis for us, I rewrote Justin's project into a simplified form that will take the updated trust .CSVs, TrustVisualizer:

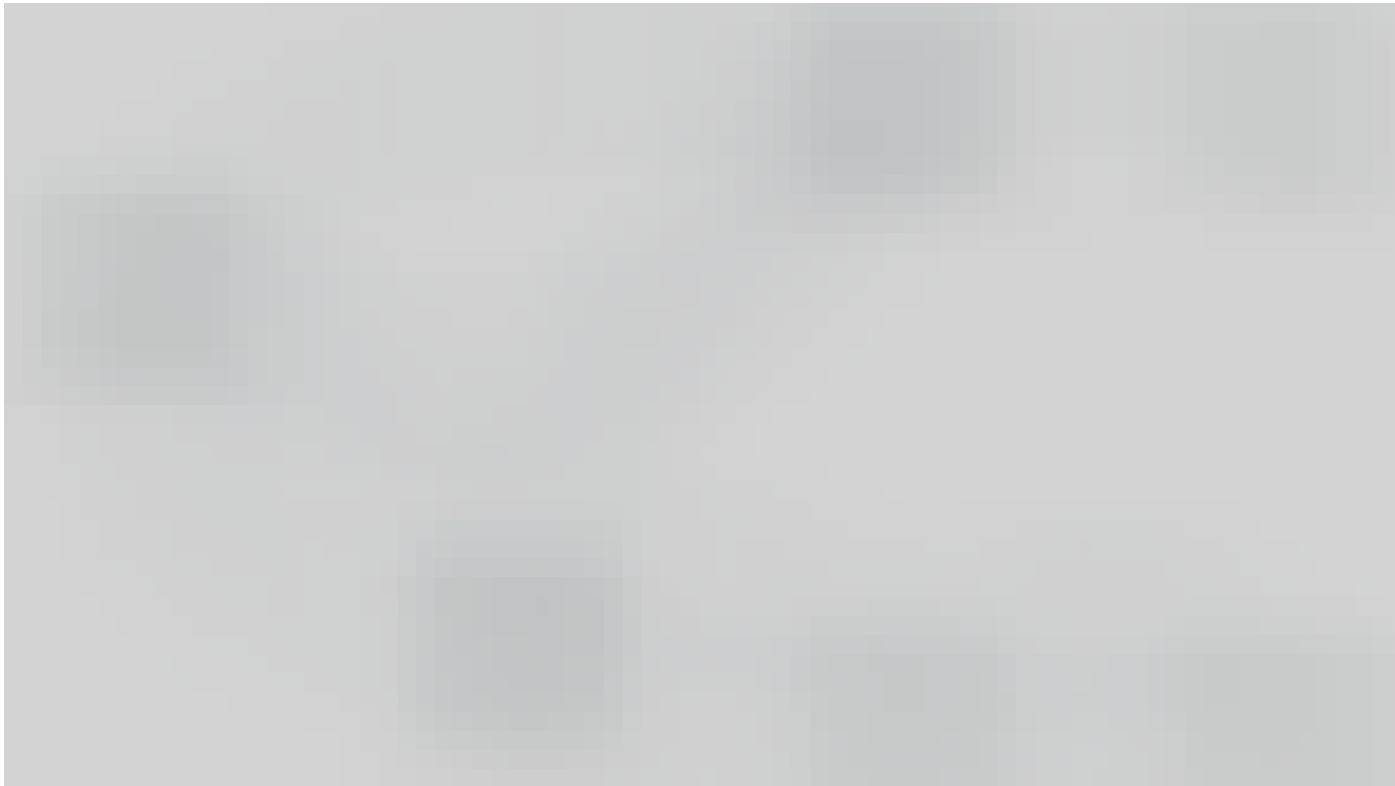


The resulting graphml can be visualized with yEd, [as described here](#). This is how I produced the the previous visualization of the sample trust architecture:



Again, with this new output, **green** edges mean “within forest”, **red** means external, and **blue** means inter-forest trust relationships. As with [@sixdub's DomainTrustExplorer](#) the edge directions for one-way trust mean *direction of access*, not *direction of trust*.

When using SharpHound to collect the trust data, and BloodHound to visualization it, here's how the same above data looks:



Foreign Relationship Enumeration

Now that we've mapped out all domain trusts reachable from the machine we're querying from, the next step in the attack planning phase hits few branches, depending on the specific types of the trusts we've encountered.

These next steps need to be executed for the hop from each each domain to another in the attack path.

If the next domain hop is in the same forest as the domain we're pivoting from/through, **and** we're able to compromise the krbtgt hash of the child domain, then we can use the method described in the following **Trustpocalypse** section to compromise the forest root.

If we're not able to compromise elevated access in the current/pivot child domain, or if the next step in the trust attack path is an external/forest trust, then we need to enumerate what users (if any) from the domain we're a part of are in groups in the target domain (or the next step in the domain attack path.)

Unfortunately, there are a lot of caveats with this step. The exact nature of the trust your current domain retains with the *trusting* domain you're querying will affect what information you can retrieve, and the exact methods you can use. In general, again, this enumeration heavily depends on whether you're querying a foreign domain in the same forest, or across an external/inter-forest trust. I'll do my best to explain all the subtleties.

There are three **main** ways that security principals (users/groups) from one domain can have access into resources in another foreign/*trusting* domain:

- They can be added to local groups on individual machines, i.e. the local “Administrators” group on a server.
- They can be added to groups in the foreign domain. There are some caveats depending on trust type and group scope, described shortly.
- They can be added as principals in an access control list, most interesting for us as principals in ACEs in a DACL. For more background on ACLs/DACLs/ACEs, check out the “[An ACE Up The Sleeve](#)” *whitepaper*.

Case 1: Local Group Membership

This involves enumerating the local memberships of one or more systems through remote SAM (SAMR) or through GPO correlation. I won’t cover this case heavily here, but will note that we have had success in the past with targeted SAMR enumeration of high value servers or domain controllers for trust-hopping. The PowerView function to do this manually is **Get-NetLocalGroupMember <server>**, and BloodHound will do this all automatically for you.

Case 2: Foreign Group Membership

The group membership case gets a bit tricky. The **member** property of an Active Directory group and the **memberOf** property of a user/group object have a special type of relationship called linked attributes. I covered this in more depth in a previous post, but with linked attributes, Active Directory calculates the value of a given attribute, referred to as the back link (e.g. **memberOf** with users/groups) from the value of another attribute, referred to as the forward link (e.g. **member** with a group). The gist is that group membership is ultimately preserved within the target group itself in the **member** property, and this all gets a bit complicated over trusts. Hopefully this will make more sense shortly. Whether or not the **memberOf** property saved with a user/group object reflects their foreign group memberships depends on the nature of the trust and scoping of the foreign group they're a member in.

Here's a breakdown of the three group scopings, and which can have what type of foreign members added:

- **Domain Local Groups** can have intra-forest cross-domain users (users in the same forest as the group) added as members, as well as inter-forest

cross-domain users (foreign security principals.)

- **Global Groups** can not have any cross-domain memberships, even within the same forest. So for our purposes we can ignore these.
- **Universal Groups** can have any user in the forest as a member, but “foreign security principals” (i.e. users from forest/external trusts) can not be a part of universal groups.

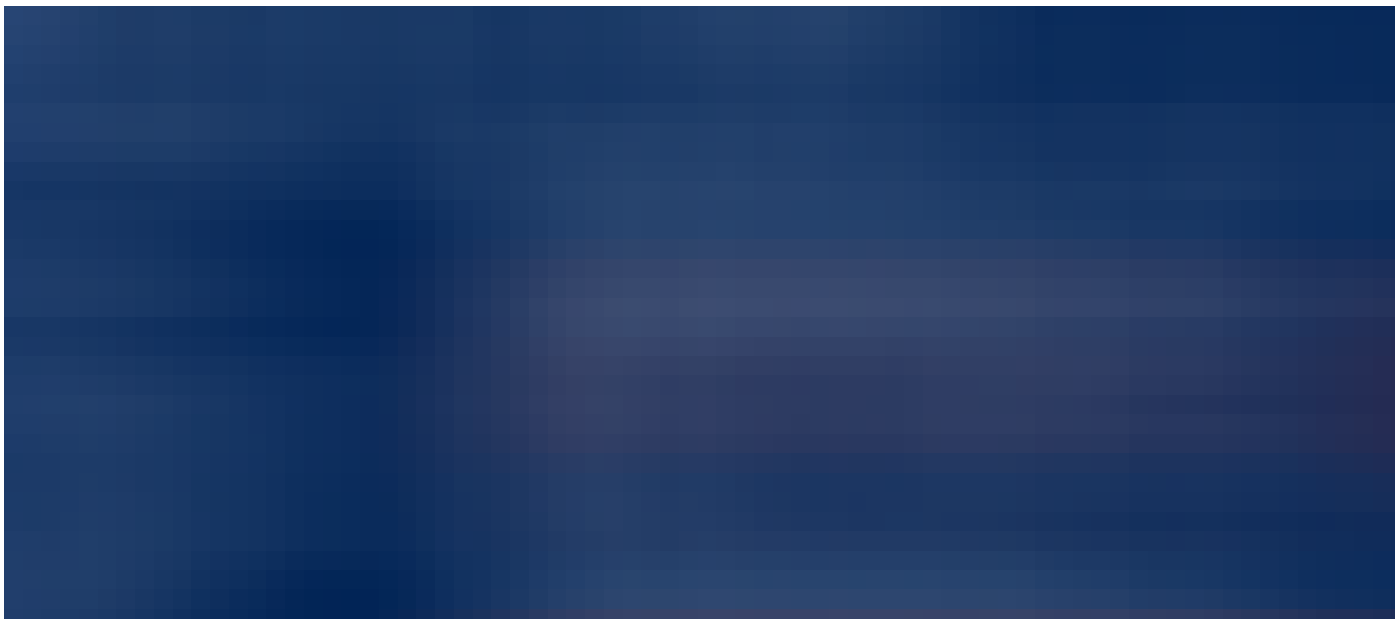
If a user/group is nested into a group in another domain that's in the same forest (so a “domain local” or “universal group”) then depending on the target group's scope the membership *might* be updated in the user/group's **memberOf** property. Groups with a “universal” scope have their memberships replicated in the global catalog for the forest, meaning a user's **memberOf** will be updated. If the group's scope the user is added to is “domain local”, then the user's **memberOf** will NOT be updated (in the global catalog), as a group with “domain local” scope does not have its memberships replicated in the forest. So the only way to tell what a user's foreign group memberships are, by solely looking at the user object, is if they are added to a universal group in the same forest. However, this also means that if we can

bind to the global catalog of a forest, we can enumerate all of these specific cross-domain relationships easily.

If the user is nested in a group in a domain over a forest/external trust, then things are treated a bit differently. Users that exist over external or forest trusts can still be added to **domain local** groups in the specified domain.

These users show up as new entries in

CN=ForeignSecurityPrincipals,DC=domain,DC=com in the domain to which they're being added, which are used as a kind of proxy that allows the foreign security identifiers to be added to resources in the domain.





As Microsoft explains it, “When a trust is established between a domain in a forest and a domain outside of that forest, security principals from the external domain can access resources in the internal domain. Active Directory creates a foreign security principal object in the internal domain to represent each security principal from the trusted external domain. These foreign security principals **can become members of domain local groups** in the internal domain”. If “domain local” or “group scoping” are foreign to you, check out my [previous post on the subject](#).

Tl;dr, as I understand it, these ForeignSecurityPrincipals act as aliases for the “real” user that’s external to the domain/forest, and it’s the ForeignSecurityPrincipal that’s actually added to groups in the target domain. The SID of a given ForeignSecurityPrincipal is the same SID as the foreign user, which makes for easy filtering later.

Case 3: Foreign ACL Principals

Luckily most of the **ntSecurityDescriptor** property of Active Directory objects is (1) accessible to any domain authenticated user, and (2) replicated in the global catalog. This means that if from your current domain context, you can query the DACLs for all objects in a *trusting* domain, and filter any ACE entries where a foreign security principal has the given right on the object you're enumerating.

You can use PowerView's **Get-DomainObjectACL -Domain <domain.fqdn>** function to retrieve these ACEs, but in order to find cross-domain DACL relationships, you will need to filter out principals/SecurityIdentifiers that do not match the SID of the domain you're querying. I'll cover this in a future **PowerView PowerUsage** post.

Operational Guidance

Note: I'll also walk over all steps needed in the **Case Study** section later in the post in case parts of this don't make sense.

If you're currently within a child domain within a forest, and *DO* have elevated access in said child domain, refer to the **Trustpocalypse** section.

If you're currently within a child domain within a forest, and *DO NOT* have elevated access in said child domain, then you can run PowerView's **Get-DomainForeignUser** function to enumerate users who are in groups outside of the user's current domain. This is a domain's "outgoing" access, i.e. users/groups who may have some kind of access into other domain groups *within the same forest*. This function can be useful to also map other intra-forest domain user/group relationships:



If you're targeting an external/forest domain, or a target domain within the same forest, you can use PowerView's **Get-DomainForeignGroupMember -Domain <target.domain.fqdn>** function. This enumerates *groups* in the target domain that contain *users/groups* who are not in the target domain.

This is a domain's “incoming” access, i.e. groups in target domain with inbound membership relationships:



Also, luckily for us, ForeignSecurityPrincipals are replicated in the global catalog, just like trusted domain objects (mentioned in the **Mapping Domain**

Trusts section). So if you want to quickly enumerate all foreign security principals (i.e. any inbound foreign groups/users) that are members of groups within a domain within the current/target forest, you can query any global catalog with an LDAP filter of **‘(objectclass=foreignSecurityPrincipal)’**. And since these foreign principals can only be added to groups with a domain local scope, we can extract the domain the foreign user was added to from the distinguishedname, query that domain directly for domain local-scoped groups with members, assuming we have some type of direct or transitive trust with that target domain. This allows us to compare the membership of these domain local groups each against the list of foreign users:

```
1 $ForeignUsers = Get-DomainObject -Properties objectsid,distinguishedname -SearchBase "GC://
2 $Domains = @{}
3
4 $ForeignMemberships = ForEach($ForeignUser in $ForeignUsers) {
5     # extract the domain the foreign user was added to
6     $ForeignUserDomain = $ForeignUser.SubString($ForeignUser.IndexOf('DC=')) -replace 'DC='
7     # check if we've already enumerated this domain
8     if (-not $Domains[$ForeignUserDomain]) {
9         $Domains[$ForeignUserDomain] = $True
10        # enumerate all domain local groups from the given domain that have any membership
11        Get-DomainGroup -Domain $ForeignUserDomain -Scope DomainLocal -LDAPFilter '(member=
12        # check if there are any overlaps between the domain local groups and the forei
```

```
13         if ($($_.member | Where-Object {$ForeignUsers -contains $_})) {  
14             $_  
15         }  
16     }  
17 }  
18 }  
19  
20 $ForeignMemberships |
```

gc_forei-

This quickly gives us a mapping of all the foreign user/group nested relationships inbound into our current (or target) forest.

If you are using BloodHound with its new SharpHound ingestor, you can still use **-Domain <domain.fqdn>** with the ingestor combined with the **-CollectionMethod** options of 'Group', 'LocalGroup', and/or 'ACL'.

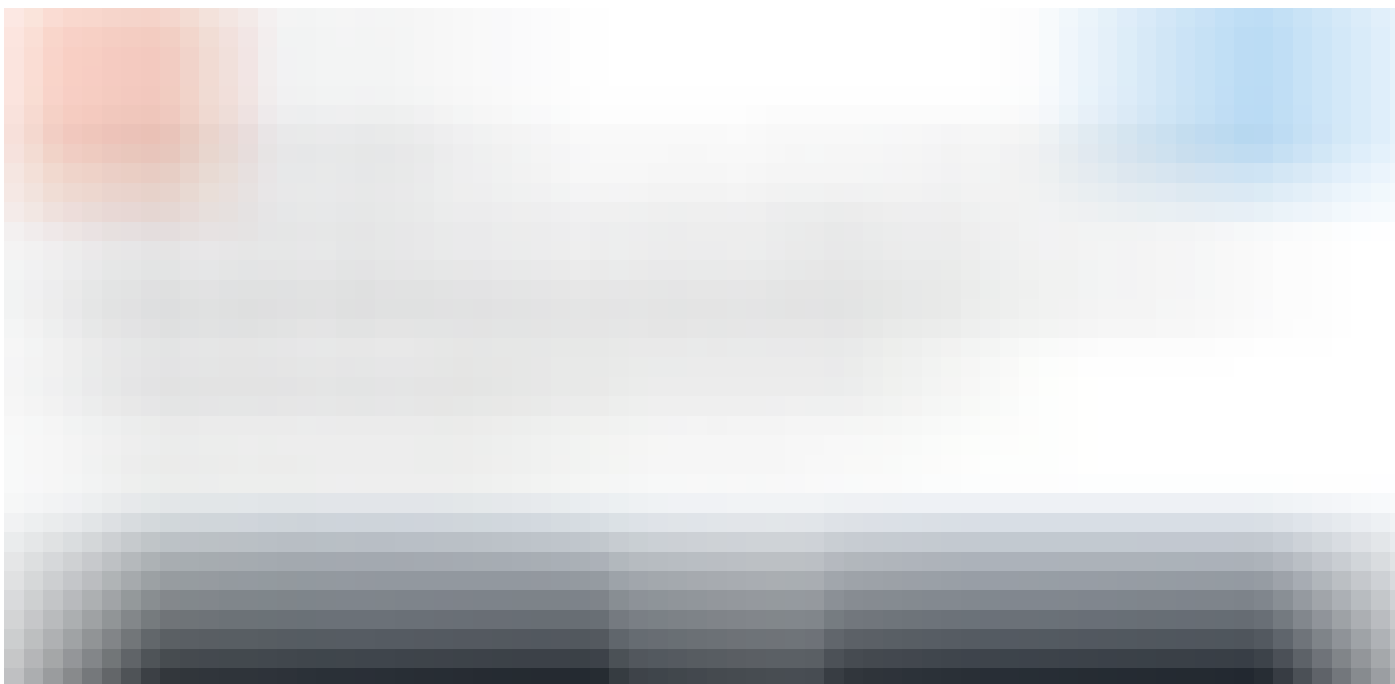
BloodHound models user/group nodes with the **name@<domain.fqdn>** syntax in the schema. This removes the requirement of having to perform complex analytics to extract these relationships after the data has been collected. If **user@dev.testlab.local** is a member of **group@testlab.local**, that **memberOf** relationship is automatically modeled. If that nested group relationship shows up in any attack paths, it will be automatically included in your graph with no extra effort.

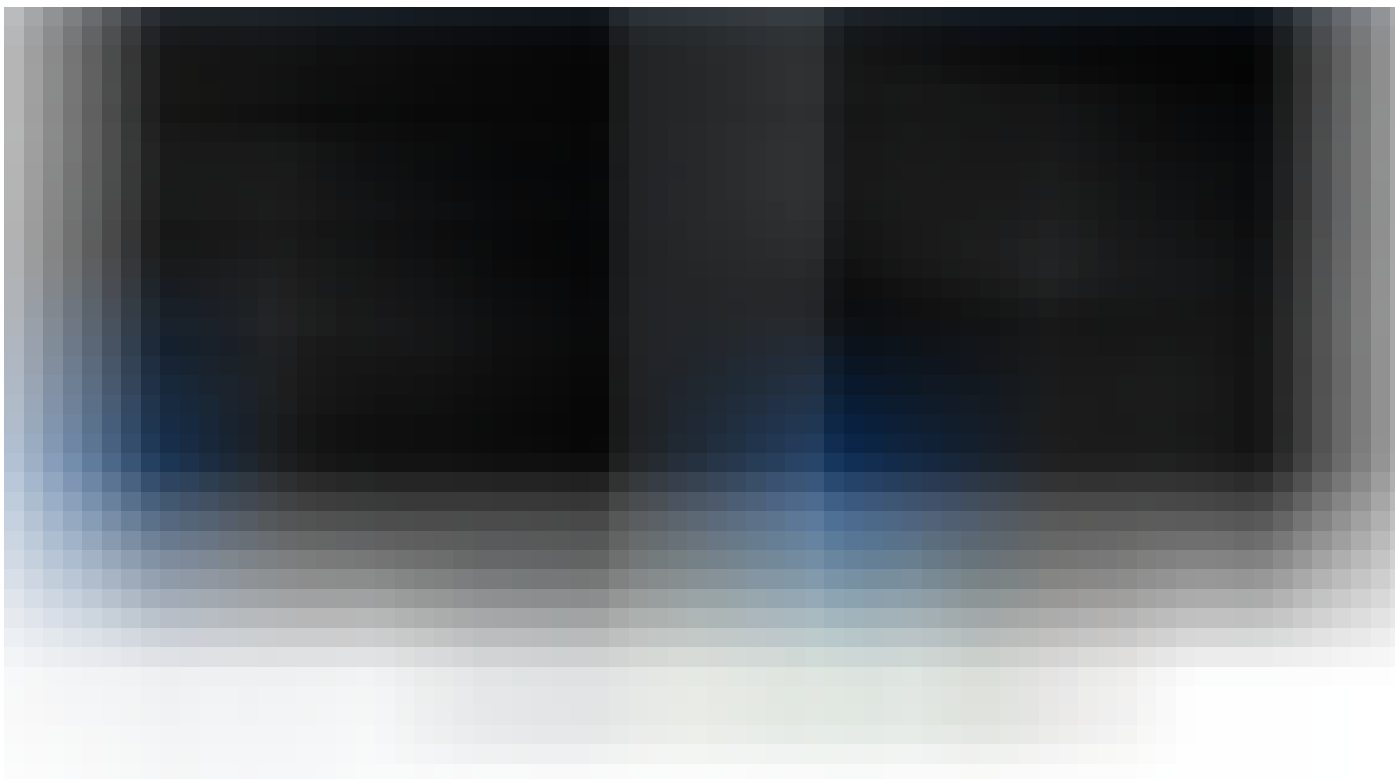
Makes perfect sense, right? :) This is a complex topic if you're not familiar, so reread the previous section a few times until it makes sense how to tease out these cross-domain relationships. Check out the **Case Study** section for a realistic walk through with the reference architecture I've used throughout this post.

The Trustpocalypse—SID Hopping Up Intra-Forest Trusts

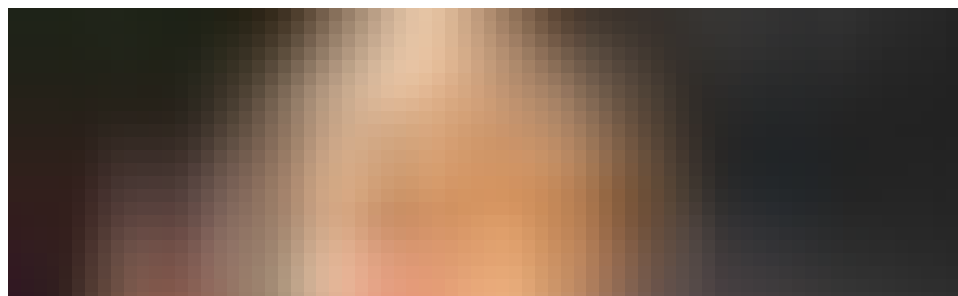
This is one of my favorite things I've learned about in the last few years in security. Just as most people remember the first time they saw Mimikatz extract a plaintext password out of memory, the memory of when I realized what this attack entailed is seared into my mind.

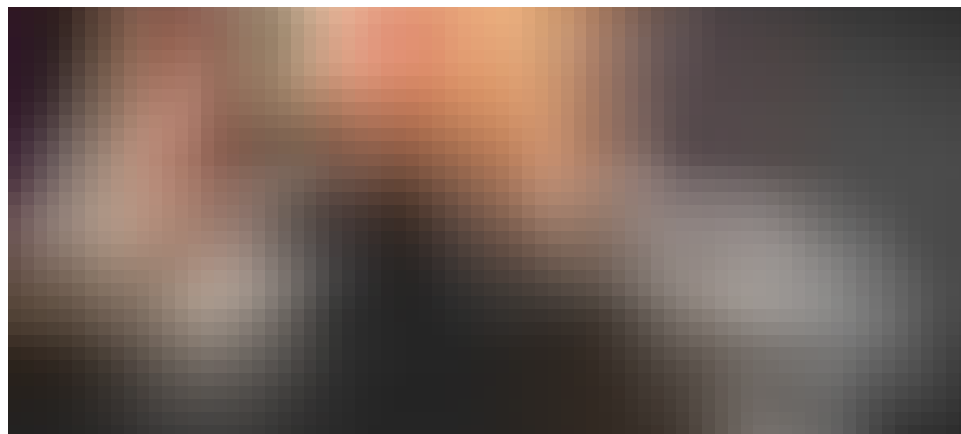
It started as many of my brain-blowing moments have, by viewing a tweet from Benjamin Delpy in June of 2015, and at first not understanding the implications:





After chatting with Benjamin to confirm what I thought the implications were, his response was “Sorry for your head :)”





This is all thanks to work that [Benjamin](#) and [Sean Metcalf](#) worked on to make Golden tickets even more “golden”. I blogged about this back in August of 2015 after their work was released in a post titled “[The Trustpocalypse.](#)”

Previous to this work, our strategy was to map out foreign user/group memberships and hop from child trust up to the forest root “by hand”, often a painstaking process in large environments with lots of domains. As described in the **Trust Attack Strategy** section, we always interpreted trust as “flowing down” from the forest root to child domains due to the “Enterprise Admins” group. However, Microsoft has stated for years that “*[the forest is the security boundary for Active Directory](#)*“, and an attack against intra-forest domains has been known since (at least) 2005.

But first, in order for this to make complete sense, I have to explain **sidHistory** and the SID filtering security mechanism, and what this all means for domains within a forest.

sidHistory was added with Windows 2000 Active Directory, and was meant to facilitate the migration of users from one domain to another. If a user is migrated, their old security identifier (SID), along with the SIDs of any group they were previously a part of, can optionally be added to the **sidHistory** attribute of their new user account. When the new user attempts to access a resource, *“if the SID or the SID history matches, access to the resource is granted or denied, according to the access specified in the ACL.”* Meaning, any group/old user SID that is set in a user’s **sidHistory** property grants them access *as if they were that user or a member of those groups*.

Due to how trusts work within an Active Directory forest, the **sidHistory** property (“ExtraSids” in the PAC) is respected within the domains of a forest because those SIDs are not filtered out in cross-domain referrals by the “SID Filtering” protection. So any user in a child domain that has their **sidHistory/ExtraSids** set to, say, the “Enterprise Admins” SID (a group that exists only in the forest root) will effectively function *as if they are an*

enterprise administrator. As Microsoft has known this is an issue, and the knowledge has been public since at least this [2005 ITPro Windows article](#) and almost certainly before, **sidHistory** is a protected attribute that is extremely difficult to modify.

Previously, abuse of this involved a pretty complex process, and included modifying the **sidHistory** in the Active Directory database (ntds.dit) of the associated domain. There's more detail about exactly why/how this works in the **Epilogue: SID Filtering** section.

THIS IS WHY THE FOREST IS THE “TRUST BOUNDARY” IN ACTIVE DIRECTORY, NOT THE DOMAIN!

Benjamin and Sean realized that with the introduction of Mimikatz' Golden Tickets, an attacker could set the **ExtraSids** section of the KERB_VALIDATION_INFO structure created for the ticket (the structure that *“defines the user's logon and authorization information provided by the DC”*). The **ExtraSids** section is described as *“A pointer to a list of KERB_SID_AND_ATTRIBUTES structures that contain a list of SIDs corresponding to groups in domains other than the account domain to which the*

principal belongs” (the KERB_SID_AND_ATTRIBUTES structure is defined [here](#).)

This means that if an attacker compromises “Domain Administrator” rights (or equivalent, actually just any account that can DCSync ;) in ANY child domain in the forest for just 5 minutes, the krbtgt hash of the child domain can be retrieved, and `/sids:<sid_of_enterprise_admins_in_forest_root>` can be added to the Mimikatz constructed ticket without modifying the Active Directory database. This gives the attacker the ability to “hop up” the forest trust relationship and compromise the forest root domain.

If this is your first time hearing about this technique, as Benjamin said, “Sorry for your head. :)”

For our operational attack strategy, this means that if we are currently in a child domain and can compromise DCSync/DA access, or if we can compromise this level of access in any child domain along our attack chain, we can forgo the burdensome foreign relationship enumeration to hop up the trust to instantly compromise the forest root. We have done this in the field,

and yes, it is awesome. :) For operational advice/guidance, check out my [previous Trustpocalypse post from 2015](#).

This will only work for hopping between trusts ***within a forest***. This will not work for external or inter-forest trusts due to SID filtering, described in more detail in the **Epilogue: SID Filtering** section at the end of the post.

A Case Study

So, consider again the sample trust diagram:



Say we land an account in **external.local**. Since **sub.dev.testlab.local** *trusts* **external.local**, **external.local** can query information from **sub.dev.testlab.local**, while SUB cannot do the same to EXTERNAL. From the external context, we can query the trusts that SUB has:



But this only returns the direct trusts that **sub.dev.testlab** has with other domains (**dev.testlab.local** and **external.local**). If we can query the global catalog (not always possible) from **sub.dev.testlab** and return all domain trusts in the entire forest!



Note that because since this is a one-way, non-transitive external trust into **sub.dev.testlab.local**, we can't query the trusts that **contoso.local** has from the EXTERNAL context, as our Kerberos traffic will not be properly referred. This is what this error usually means, in case you run across it:



So, from here we would then run **Get-DomainForeignGroupMember -Domain sub.dev.testlab.local** to see if any groups in SUB contained members in EXTERNAL:





From there, we would attempt targeted account compromise to hop the trust into **sub.dev.testlab.local**. The **Get-DomainForeignUser** command would be of no use here, due to the caveats about linked-value replication and trusted described in the **Foreign Relationship Enumeration** section.

However, because **external.local -> sub.dev.testlab.local** is an external trust relationships, it is implicitly non-transitive, so couldn't query the domain local group memberships of **dev.testlab.local** or **testlab.local**.

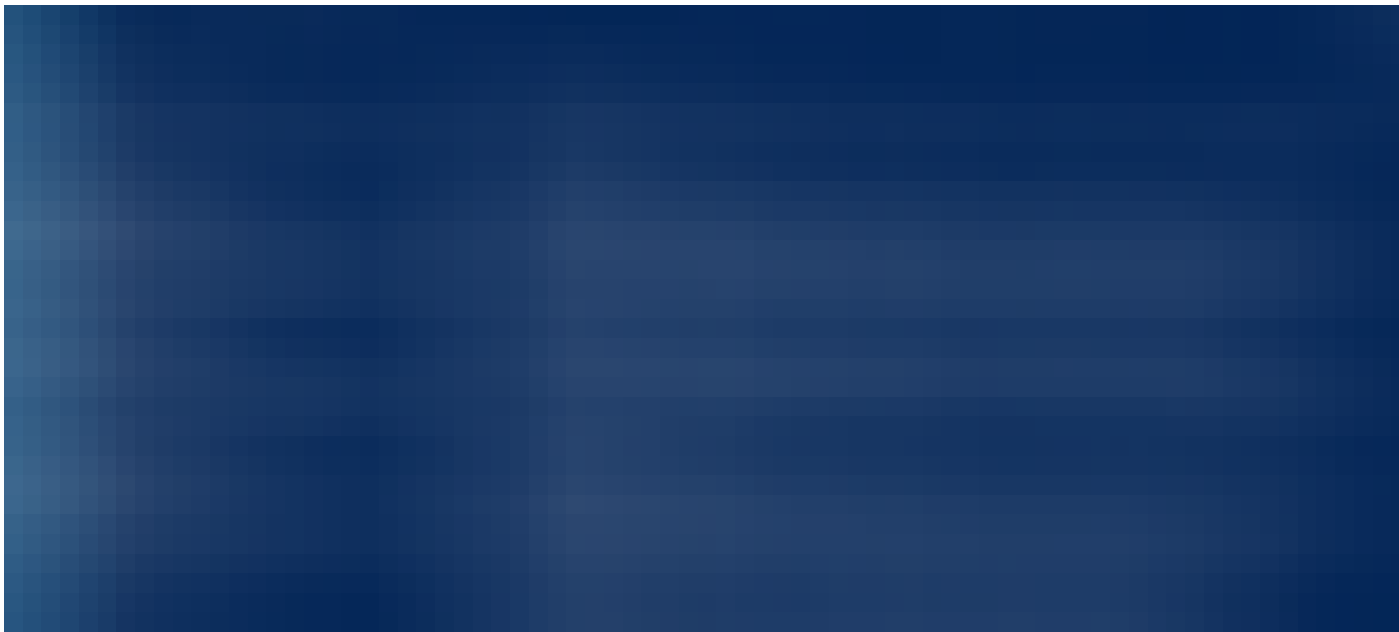
If we were then able to compromise domain admin (or equivalent) credentials in **sub.dev.testlab.local**, we could build a sidHistory-trust-hopping Golden Ticket as described in the “**Trustpocalypse**” section to compromise the **testlab.local** forest root domain. If we weren’t able to procure elevated access, we would run **Get-DomainForeignUser** to see if any users from **sub.dev.testlab.local** had access into other groups in the forest. Again, remember the previous information about scoping- only *universal* group memberships will be reflected here:



We would also run **Get-DomainForeignGroupMember -Domain dev.testlab.local** and **Get-DomainForeignGroupMember -Domain testlab.local** to see that groups in those other forest domains had “incoming” access:



Once/if we were able to compromise part or all of the **testlab.local** forest root through either of the previous approaches, we would then run **Get-DomainForeignGroupMember -Domain contoso.local** and **Get-DomainForeignGroupMember -Domain prod.contoso.local** to see if there were any users in the TESTLAB forest that had foreign group membership in the CONTOSO forest.



Along the way, we would could run **Get-NetLocalGroupMember <foreign,server>** against a targeted selection of servers (including DCs) to

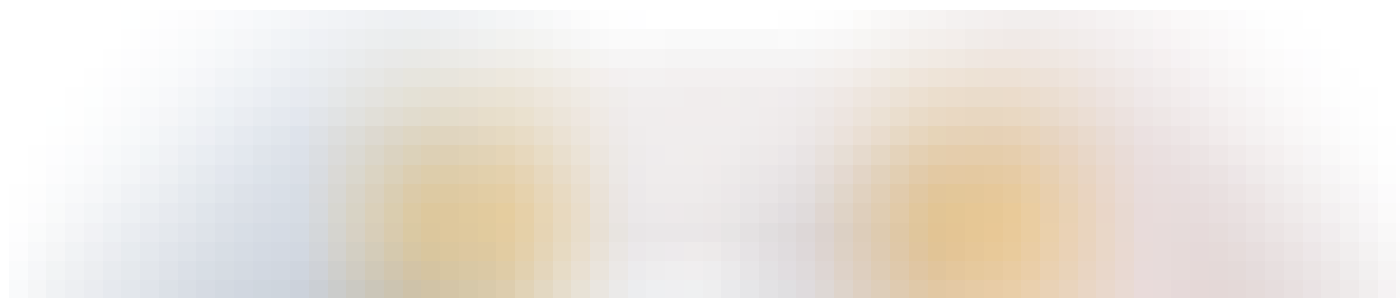
see if any users crossed the boundary that way via machine local groups. We could also use targeted **Get-DomainObjectACL -Domain <foreign.domain>** with various filters to check for foreign ACL memberships.

Or we could just pull everything with BloodHound, and rely on the schema to model the cross-forest hops. :)

Sidenote: Forging Inter-Realm Trust Tickets

It is possible to forge inter-realm trust tickets to exploit trust relationships. As Sean covered this extremely well in his [“It’s All About Trust”](#) post, I’ll refer you to his documentation for more operational details, and will just cover the implications of this technique and how it fits into our trust attack strategy.

Recall the explanation of how Kerberos works across trusts:

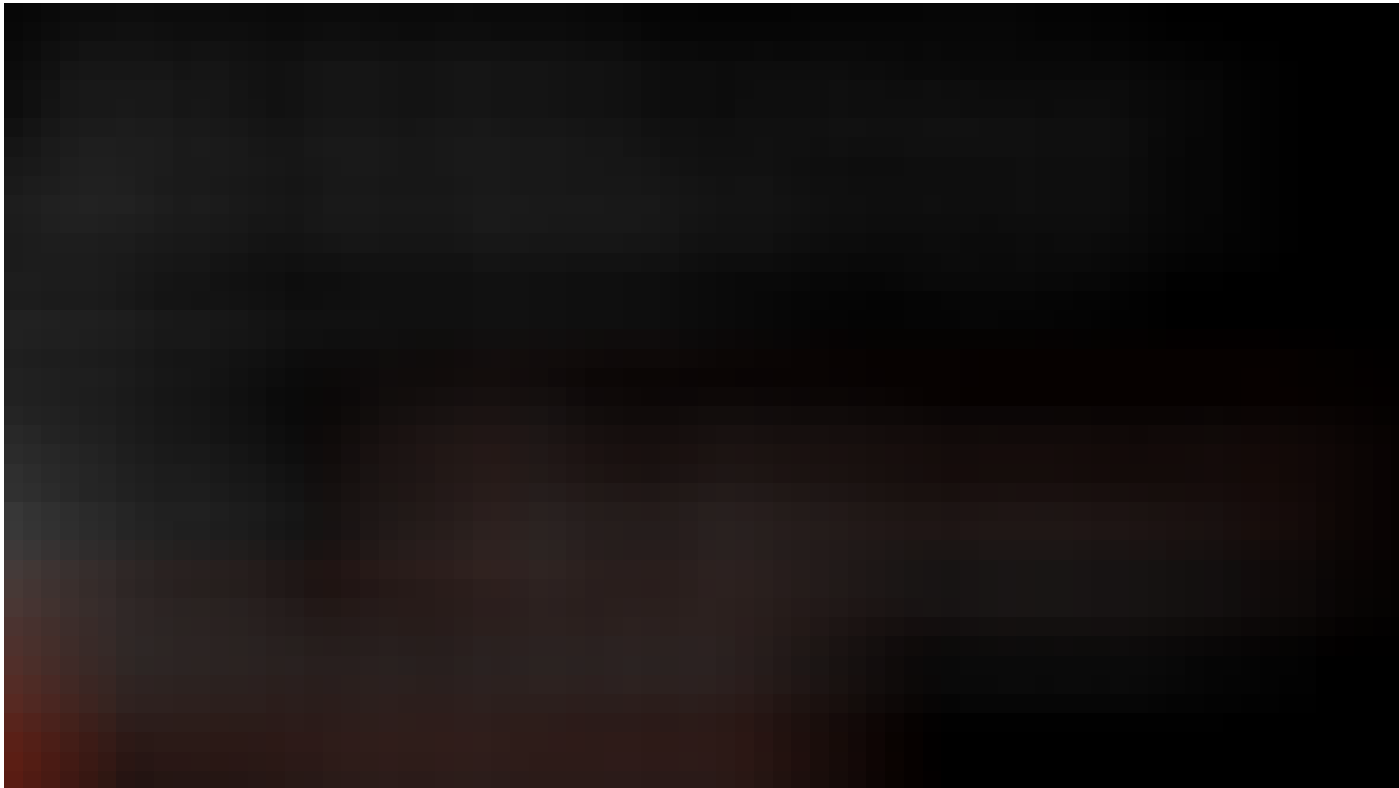




So when the user presents this inter-realm ticket-granting-ticket referral to the foreign domain, again signed by the inter-realm trust key, the user's TGT is included within it. And again, because the foreign domain trusts the domain that that issued the referral ticket, the foreign domain trusts the user's TGT and all its included information to be accurate.

Again, in English, when the foreign domain decrypts the referral ticket with the negotiated trust key, it sees the user's TGT and says *“OK, the other domain already authenticated this user and said this is who they say they are/these are the groups the user is in, so I'll trust this is accurate because I trust this domain.”*

So, if we can retrieve the hash of the inter-realm trust key, a referral ticket can be forged (as Sean describes) that allows us to pretend to be any user from the first domain when requesting access to the second domain. This hash retrieval can be done through normal password dumping or through DCSync, by querying the **FOREIGN_DOMAIN_SHORTNAME\$** account:



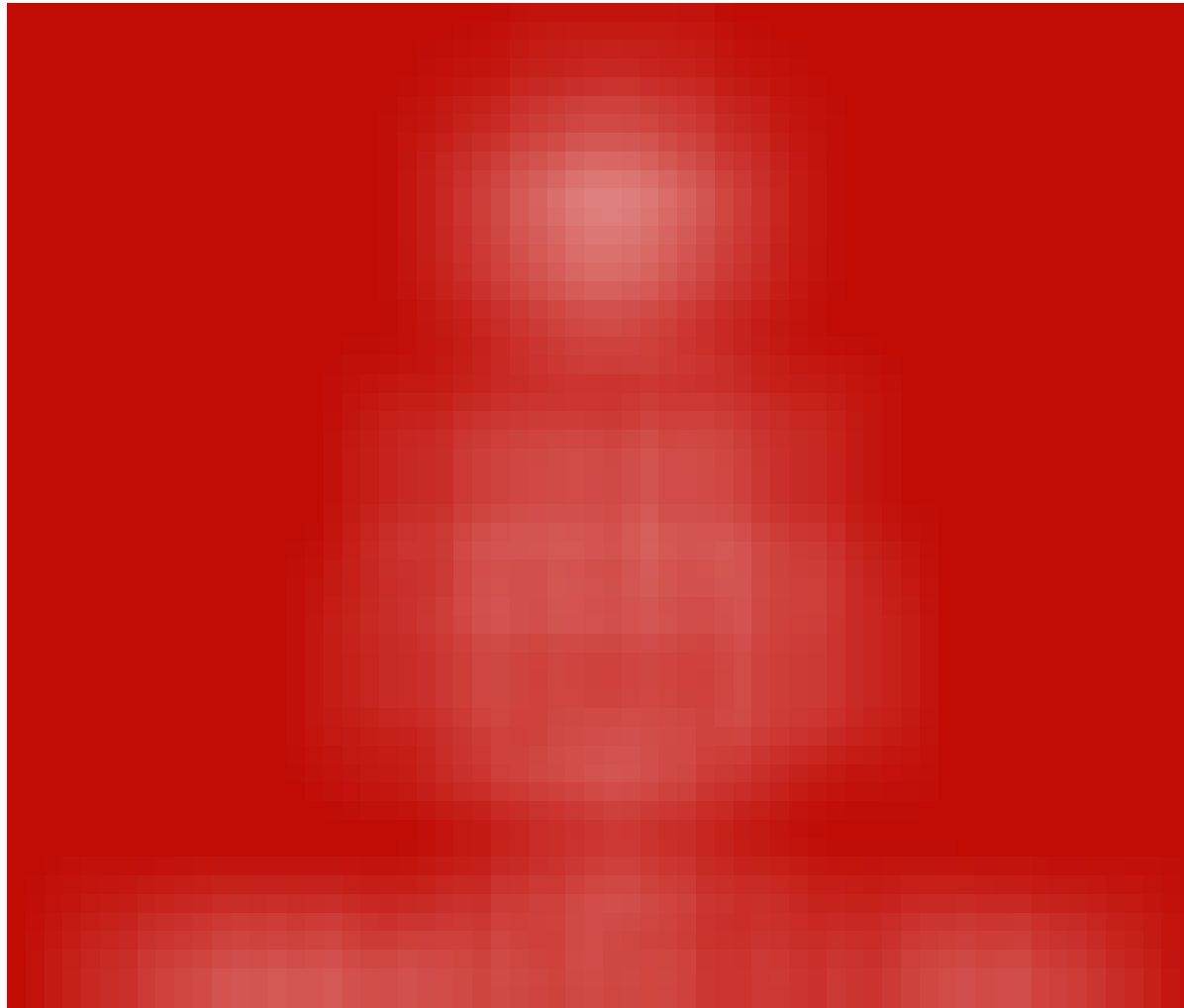
However, if we can retrieve the inter-realm trust key, then in pretty much all cases we can pull the krbtgt hash of the referring domain. If we have this, we can construct a ticket for user referring domain, pretending to be any user we want to the foreign domain. This is why I haven't had a need to forge inter-realm trust referrals in the field, but there is one specific instance where it gets interesting.

Back in 2015, after everyone started to fully realize the implications of Golden Tickets, [Microsoft released scripts](#) that allow organizations to change the password of the krbtgt account. In order for this to be effective for a single-domain forest, the password has to be changed twice. Now, because of the implementation of the sidHistory-hopping attack, the password for the krbtgt account in EVERY domain in the forest, twice.

Say an organization does this, and rotates the passwords for *every* elevated account in *every* domain in the forest, are they safe? Well, while inter-realm trust keys automatically rotate every 30 days according to section [6.1.6.9.6.1](#) of the Active Directory Technical Specification, they aren't rotated when the krbtgt account changes. So if an attacker has the inter-realm keys in their possession, they can still use the sidHistory approach to hop up a trust, as

Sean details. Then again, there's a million and one other ways to backdoor Active Directory. ^_('ツ)_/^-

So there's only one solution if you want to be sure (thanks @gentilkiwi :)





Another Sidenote: Kerberoasting Across Domain Trusts

We love Kerberoasting. Introduced by Tim Medin in 2014, we ‘roast on a ton of our engagements. And if we have a domain that trusts us, we can roast across these trust boundaries, with one minor tweak in some situations.

When using .NET’s

System.IdentityModel.Tokens.KerberosRequestorSecurityToken class (and then its .GetRequest() method), we specify a service principal name (SPN) to request a TGS-REP for, and subsequently use the GetRequest() to retrieve the bytes for the AP-REQ that’s intended to be sent to the target service. This AP-REQ contains the service ticket that we then extract and use for offline Kerberoasting/password cracking.

The documentation for the KerberosRequestorSecurityToken.ServicePrincipalName nicely describes the format as “host/<hostname>@<domain> or <hostname>, where hostname is the name of the computer hosting the target Web service and domain is the fully-qualified domain name of the Kerberos realm in which the host computer resides.” So if you have any issues with Kerberoasting across trusts (particularly external and forest trusts), try using the **SERVICE/host.domain.com@domain.com** format and you may have more success. This is possible with PowerView’s **Get-DomainSPNTicket**, the function that **Invoke-Kerberoast** is built on.

Epilogue: SID Filtering

So we previously talked about how/why the forest is the trust boundary in Active Directory, not the domain. A big part of this is the security protection I’ve alluded to several times previously called SID Filtering. The best reference for SID filtering is the [MS-PAC] “Privilege Attribute Certificate Data Structure” documentation, specifically section 4.1.2.2 “SID Filtering and Claims Transformation.” I will do my best to explain a few salient points.

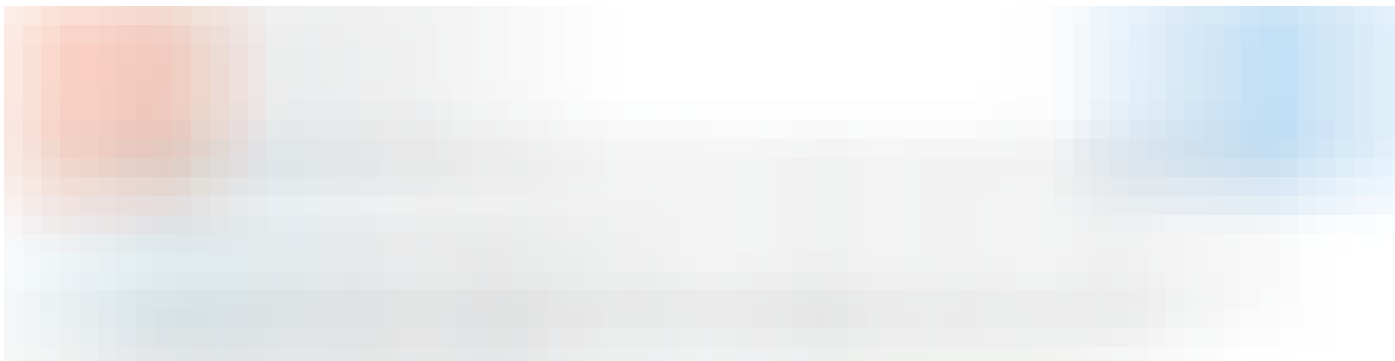
When a user's TGT is presented to the new domain through a referral, that TGT contains a privileged attribute certificate (PAC) that contains, among other things, the user's security identifier (SID), the security identifiers of groups they are in, and anything present in the previously discussed sidHistory field (i.e. the **ExtraSids** PAC part described in the **Trustpocalypse** section). This security identification information in the PAC is parsed and analyzed by a *trusting* domain, and various filters are executed depending on the type of the trust.

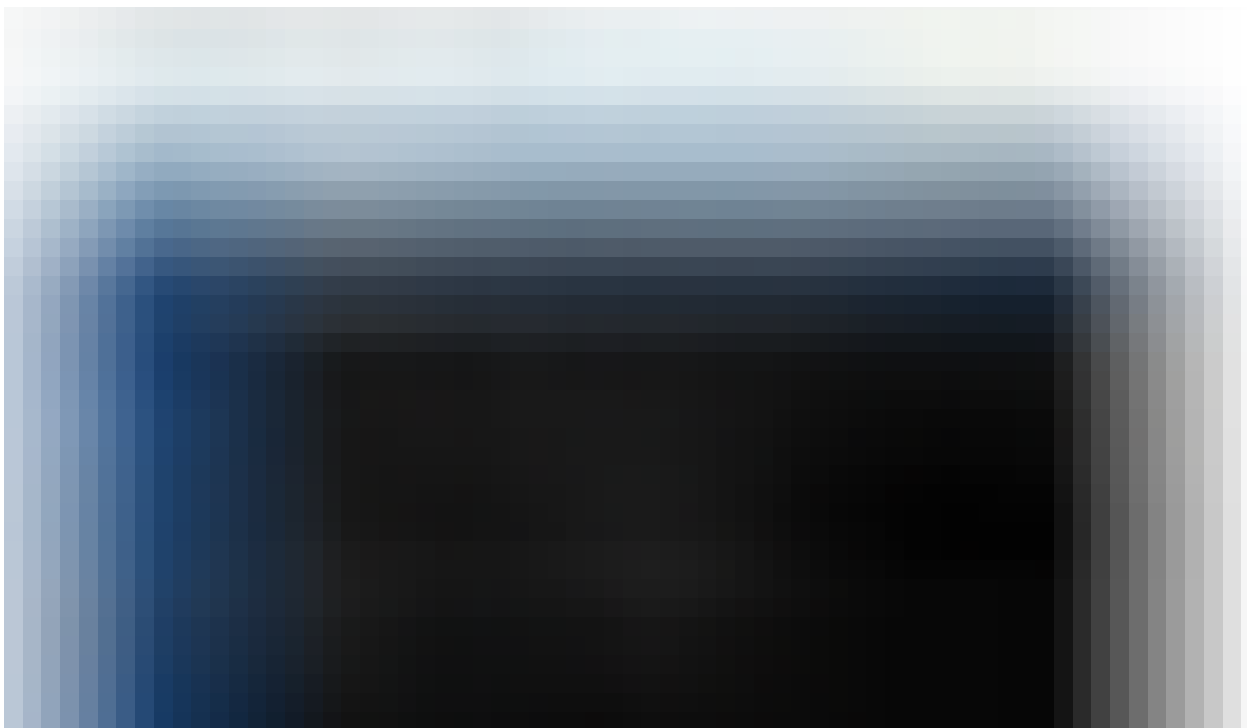
SIDs matching particular patterns are rejected by the trusting domain under various circumstances, as a security protection. SID filtering is meant to stop malicious users with elevated credentials in a *trusted* domain/forest from taking control of a *trusting* domain/forest. This is also described in Microsoft's "Security Considerations for Trusts" documentation.

There is a set of SIDs that are set to 'AlwaysFilter', meaning they are always filtered out by a trusting domain, no matter the trust type. The main SID we're interested in, "Enterprise Admins" (S-1-5-21-<Domain>-519), the one that allows us to execute the sidHistory-hopping attack, is set to "ForestSpecific" for filtering. As Microsoft describes, "The ForestSpecific rule is for those SIDs

that are never allowed in a PAC that originates from out of the forest or from a domain that has been marked as QuarantinedWithinForest, unless it belongs to that domain.” Again, this explains why the forest is the trust boundary, not the domain, as this elevated SID (along with many others) can not be passed across a trust boundary except if the target domain is within the same forest.

QuarantinedWithinForest, huh? It so happens that domains *within* a forest can be set as “quarantined”, which implements a version SID filtering for the domain, even though it is within the forest. However, as the documentation states, *“The only SIDs that are allowed to be passed from such a domain are the “Enterprise Domain Controllers” (S-1-5-9) SID and those described by the trusted domain object (TDO).”* So, since the “Enterprise Domain Controllers” SID is NOT filtered out for intra-forest quarantined domains, there is still a way to “hop up” the forest trust chain and compromise the forest root:





This is something I attempted to explain a few years ago without properly understanding the problem, but it makes a bit more sense now after reading heaps of Microsoft documentation :)

Wrapup

Trusts are not a simple topic. Most pentesters (and many sysadmins!) do not properly understand trusts and the risk exposed by various trust misconfigurations. Unfortunately for us, some bad guys do, and trusts have

been abused since nearly the beginning of Active Directory to exploit access in one domain in order to pivot into another.

If your domain trust architecture is setup incorrectly, then what's the answer? Unfortunately, as with many major architectural flaws of this nature, there is not a simple fix. Rearchitecting a major Active Directory deployment can be a long, expensive, and painful process, but there is still a guiding light: the Enhanced Security Administrative Environment (ESAE), commonly referred to as “Red Forest”, which is a secured Active Directory architecture that mitigates an enormous number of Active Directory vulnerabilities/misconfigurations. While Microsoft has published aspects of the architecture, the only way that I currently know how have an ESAE/Red Forest is to pay Microsoft to implement it for you. ^_('▽)_/-

I would love to eventually publish guidance on how organizations implement a “red-forest-esque” environment, that while not “officially” compliant with the reference architecture, would still be better than many organization's current implementations. If anyone has documentation or guidance on how to do practically do this, please contact me (@harmj0y or will [at] harmj0y.net) and I will happily get that kind of information out there.

. . .

Originally published at [harmj0y](#).

Domain Trusts

Powerview

Active Directory

One clap, two clap, three clap, forty?

By clapping more or less, you can signal to us which stories really stand out.

138



Will

Follow

Co-founder of Empire/BloodHound/Veil-Framework | PowerSploit developer | Microsoft PowerShell MVP | Security at the misfortune of others | <http://specterops.io>



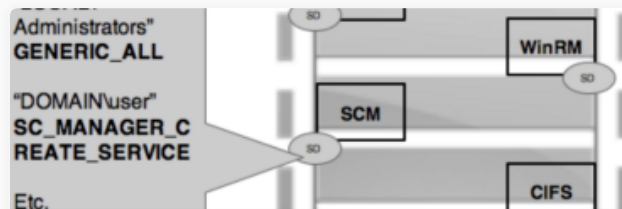
Posts By SpecterOps Team Members

Follow

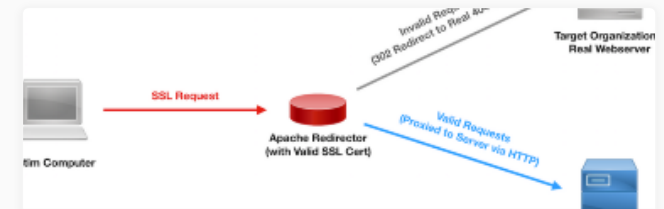
Posts from SpecterOps team members on various topics relating information security

More from Posts By SpecterOps Team Members

A Push Toward Transparency



More from Posts By SpecterOps Team



More from Posts By SpecterOps Team

Information security is a young field, which is

David McGuire
6 min read



32



Members

Remote Hash Extraction On Demand Via Host Security...



Will
13 min read



60



Members

HTTPS Payload and C2 Redirectors



Jeff Dimmock
10 min read



40



Responses



Write a response...