Hochschule Kempten
Fakultät Informatik

Bachelorarbeit

Recherche und Analyse ausgewählter IT-Sicherheitslücken im zeitlichen Umfeld von 2018

Johannes Börmann

Gutachter(in):	Prof. Dr. Elmar Böhler; Prof. Dr. Stefan Frenz
Verfasser:	Johannes Börmann
Matrikel-Nr.:	326925
Adresse:	Marktplatz 6, 87671 Ronsberg
E-Mail:	johannes.f.boermann@stud.hs-kempten.de
Eingereicht:	31.05.2019

Kurzzusammenfassung

In dieser Arbeit werden zwei IT-Sicherheitslücken, welche im zeitlichen Umfeld von 2018 bekannt geworden sind, analysiert. Die erste Sicherheitslücke führte dazu, dass sich die Erpressungssoftware, bekannt unter dem Namen “WannaCry“, verbreiten konnte. Dabei handelt es sich um fehlerhafte Implementierungen im Netzwerkprotokoll SMB von Windows, durch welche eine “Remote Code Execution“ ermöglicht wurde. Die zweite Sicherheitslücke führte zu der bisher größten, öffentlich bekanntgewordenen, DDoS-Attacke und kann unter der Bezeichnung “UDP-based amplification attack“ eingeordnet werden. Hierbei handelt es sich um eine Ausnutzung der Software “Memcached“. In beiden Fällen werden hierzu jeweils die technischen Hintergründe analysiert, um so an ein detailliertes Verständnis dieser zwei Sicherheitslücken zu gelangen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Relevanz von Themen der IT-Sicherheit hat in den letzten Jahren einen deutlichen Zuwachs erfahren können. Schlagworte wie “Cyberwar“ und “Cyberangriff“ sind heutzutage immer öfter in den Nachrichten bzw. Medienberichten zu finden und haben es schon seit längerer Zeit bis in die Politik geschafft. Spätestens mit den Enthüllungen von Edward Snowden und der daraus entstanden NSA-Überwachungsaffäre wurden diese Themen, auch aus politischer Sicht, in das Rampenlicht gerückt.
Fast alltäglich begegnet man, zumindest in gängigen IT-Nachrichtenjournalen, in diesem Zusammenhang auch dem Begriff der “Sicherheitslücke“. Die zunehmende Vernetzung der Geräte (Stichwort: Internet of Things (IoT)) führt dazu, dass für diesen Einsatzzweck immer mehr Software benötigt und entwickelt wird. Dabei birgt jedes entwickelte Stück Software natürlich die Gefahr, eine oder mehrere Sicherheitslücken zu beinhalten. Neben den zwei in dieser Arbeit analysierten Sicherheitslücken lässt sich auch an Beispielen wie *Spectre und Meltdown*, sowie *Stuxnet* sehr gut ableiten, welche gravierenden Folgen sich aus IT-Sicherheitslücken ergeben können. Durch Stuxnet konnte man deutlich erkennen, wie sich durch die Ausnutzung einer solchen Sicherheitslücke, sogar erhebliche Auswirkungen auf die physikalische Welt (in diesem Fall Urananreicherungsanlagen) abbilden konnten. Andererseits sind Spectre und Meltdown ein sehr treffendes Beispiel für Sicherheitslücken, welche grundlegende Systemkomponenten (Prozessoren) betreffen, die weltweit sehr häufig im Einsatz sind. Beide diese Beispiele verdeutlichen noch einmal, wie wichtig das Thema IT-Sicherheit, vor allem in der heutigen Zeit, einzuschätzen ist. Dies zeigt sich beispielsweise auch daran, dass Institutionen in Deutschland, wie die Bundeswehr und der Bundesnachrichtendienst (BND), ihr Augenmerk vermehrt auf Themen der IT-Sicherheit legen.
In dieser Arbeit werden zwei ausgewählte Sicherheitslücken, im zeitlichen Umfeld von 2018, näher untersucht. Dabei soll es nicht nur das Ziel sein, über diese Sicherheitslücken und deren Auswirkungen zu berichten, sondern dem Leser ein genaues Bild über die technischen Details der jeweiligen Sicherheitslücke zu vermitteln. Im Rahmen der Recherche zu dieser Arbeit ist deutlich geworden, dass zwar grundlegend über Sicherheitslücken berichtet wird, aber es allerdings oftmals an technischen Details zu den Hintergründen fehlt. Wenn detailliertere Berichte (Deep Dives, Proof of Concepts) exis-

tieren sollten, dann werden diese meist entweder von Privatpersonen (z.B. in Blogs) oder spezialisierten IT-Sicherheitsunternehmen veröffentlicht. Dies führt dazu, dass die Informationen schwer aufzufinden sind und dadurch vermutlich auch einer breiteren Masse verwehrt bleiben. Jedoch ist es im IT-Bereich von essentieller Bedeutung sich auch mit den technischen Details von Sicherheitslücken und den daraus entstandenen Exploits in der Vergangenheit auseinanderzusetzen, wenn man in der Zukunft darauf reagieren möchte. Nur mit diesem Wissen kann man sich ein Bild davon machen, wie Angriffe im Detail funktionieren, um so in der Zukunft entsprechende Gegenmaßnahmen einleiten zu können. Außerdem verhelfen diese Informationen dazu, die Denkweise und Arbeit der Angreifer nachvollziehen zu können. Dies ist insbesondere bei der eigenen Entwicklung von Software von Vorteil, um diese von Anfang an sicherer zu gestalten. Das Ganze ist vergleichbar mit der Arbeit eines Polizeiermittlers, der vergangene Verbrechen genau analysiert, mit dem Ziel, diese Art von Verbrechen in der Zukunft verhindern zu können. Um die Wichtigkeit von IT-Sicherheit insgesamt noch einmal zu verdeutlichen, wurden für diese Arbeit Sicherheitslücken ausgewählt, welche ernstzunehmende Auswirkungen in der Vergangenheit hatten.

Hinweise zum Lesen: Im Laufe der Zeit hat sich für IT-Sicherheitslücken der Standard mit der Bezeichnung "CVE" (Common Vulnerabilities and Exposures) entwickelt. Dieser dient dazu, bekanntgewordene Sicherheitslücken einheitlich zu definieren. Die Nummerierung erfolgt dabei nach diesem Schema:

CVE - $\underbrace{2018}_{\text{Jahr}}$ - $\underbrace{0012}_{\text{fortlaufende Nummer pro Jahr}}$

Die beiden Kapitel der Sicherheitslücken in dieser Arbeit können vollkommen unabhängig voneinander gelesen werden. Es wurde jedoch darauf geachtet, einen ähnlichen Aufbau zu verwenden. Zur besseren Einordnung wird am Anfang jedes Kapitels eine Hinführung zu der Sicherheitslücke gegeben, in welcher bekanntgewordene Ausnutzungen, anhand eines Beispiels, näher erläutert werden. Anschließend folgt ein Grundlagenabschnitt, durch welchen man einige wichtige Wissensvoraussetzungen erhält, um die spätere Analyse besser nachvollziehen zu können. Die darauffolgende Analyse beinhaltet die Schwerpunkte der Arbeit zu der jeweiligen Sicherheitslücke. Abgeschlossen wird jedes Kapitel durch ein gezogenes Fazit.

2 Analyse der Sicherheitslücken

2.1 CVE-2017-0144: Windows SMB Remote Code Execution Vulnerability

2.1.1 Hinführung

In diesem Kapitel wird eine Sicherheitslücke analysiert, dessen Ausnutzung im Jahr 2017 zu einem sehr hohen Maß an öffentlichem Aufsehen beigetragen hat. Hierbei handelt es sich um eine Lücke, die mit Hilfe des Exploits "EternalBlue", Schwachstellen in der SMB Implementierung von Windows ausnutzt (der Begriff "EternalBlue" dient in dieser Arbeit sowohl als Bezeichnung der Sicherheitslücke, sowie des zugehörigen Exploits). Dieser Exploit ist in der Lage einen Prozess anzustoßen, mit dessen Hilfe sich der Wurm *WannaCry* auf schätzungsweise über 300.000 Rechnern in ca. 150 Ländern verbreiten konnte [19]. Nicht nur Privatanwender, sondern auch größere Einrichtungen bzw. Unternehmen, wurden von der Attacke schwer getroffen. Darunter zählen beispielsweise der National Health Service von England, verschiedene Krankenhäuser in den USA oder auch die Deutsche Bahn hier in Deutschland [11].
WannaCry lässt sich unter der Rubrik "Ransomware" einstufen. Der Begriff Ransomware steht für eine Art von Schadprogrammen, die den Zugriff auf Daten und Systeme einschränken oder unterbinden. Entweder sperrt ein solches Schadprogramm den Systemzugriff, sodass sich beispielsweise Programme auf einem PC nicht mehr aufrufen lassen, oder es verschlüsselt bestimmte Nutzerdaten [13]. Im Fall von Wannacry handelt es sich um Letzteres.
Die Ransomware verschlüsselt auf den infizierten Rechnern alle Dateien, die einem bestimmten Dateiformat entsprechen (176 Dateiendungen insgesamt [32]). Darunter fallen beispielsweise Dateien im Microsoft Office Umfeld, Dateiarchive, Multimedia uvm. Über ein Programm wurden die Opfer dieser Attacke dazu aufgefordert, einen Betrag von 300 Dollar in Form von Bitcoins an ein bestimmtes Wallet zu übersenden, um eine Entschlüsselung der Dateien zu erwirken.
Das "Entschlüsselungsprogramm" bzw. die grafische Komponente der Schadsoftware ist in Abbildung 2.1 dargestellt.

Abbildung 2.1: Wana Decrypt0r 2.0 [9]

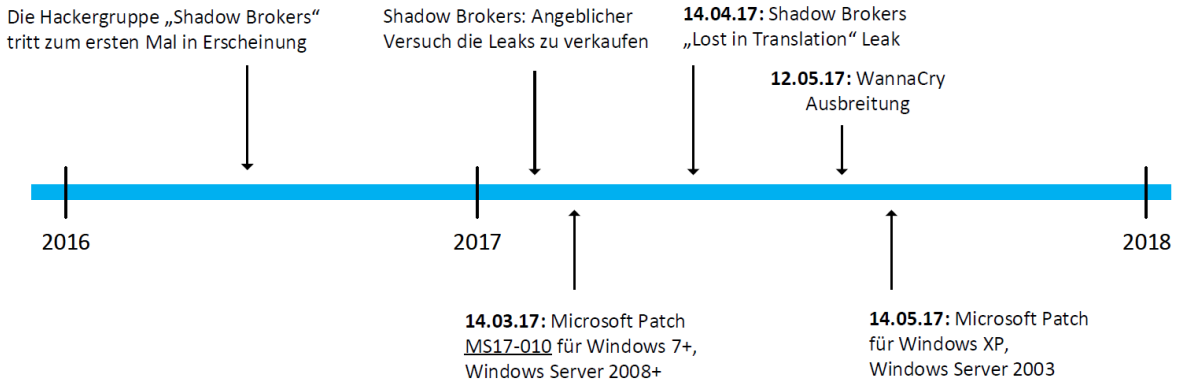


Abbildung 2.2: Zeitlicher Verlauf bis zur Verbreitung von WannaCry

Zeitliche Entwicklung Die Hackergruppe “Shadow Brokers“ ist 2016 das erste Mal aktiv in Erscheinung getreten und ist für mehrere Leaks aus den Exploits der NSA verantwortlich [25]. Es wird vermutet, dass diese Gruppe Anfang 2017 versuchte ihre neuesten Leaks (darunter auch EternalBlue) an einen Abnehmer zu verkaufen [27]. Da sich kein Käufer finden lies, entschied sich die Gruppe dazu, die Leaks am 14.04.2017 gebündelt der Öffentlichkeit zur Verfügung zu stellen. Auffallend ist hierbei, dass Microsoft jedoch schon einen Monat zuvor, am 14.03.2017, bereits einen Patch bereitstellen

konnte, der die Sicherheitslücke in der SMB Implementierung für aktuelle Betriebssysteme schließt (Abbildung 2.2). Verschiedenen Aussagen von ehemaligen und aktuellen Offiziellen der NSA zufolge, hatte der Geheimdienst Microsoft vor dem Exploit gewarnt, als sie mitbekommen hatten, dass ihre Tools entwendet wurden [22]. Dieser Umstand würde erklären, wie Microsoft mit dem Patch, den veröffentlichen Leaks zuvor kommen konnte. Schätzungen zufolge soll die NSA schon seit ca. fünf Jahren von den Sicherheitslücken und dem dazugehörigen Exploit EternalBlue gewusst haben [22]. Mit dem Leak wurde der NSA ein komplettes Framework (Name: FuzzBunch) entwendet, welches ähnlich wie Metasploit, dazu dient, anwenderfreundlich Exploits auszuführen. In diesem Framework befindet sich unter anderem auch eine ausführbare Datei mit dem Namen "EternalBlue2.2.0.exe".

2.1.2 SMB Allgemein

Da es sich bei dieser Sicherheitslücke um eine fehlerhafte Implementierung des SMB Protokolls handelt, sollen in diesem Kapitel zunächst einige grundlegende Informationen zu dessen Funktionsweise erläutert werden.
SMB (Server Message Block) ist ein Protokoll, das in Netzwerken für die Übertragung von bestimmten Daten zwischen einem Client und einem Server verantwortlich ist. Es wird verwendet, um in Windows-Netzwerken Datei- und Verzeichnisfreigaben sowie Druckdienste zu realisieren [2]. Um auch von Unix Systemen aus solche Dienste nutzen zu können bzw. um "gemischte" Netzwerke realisieren zu können, wurde für diesen Zweck das Programmpaket "Samba" entwickelt.
SMB wurde 1983 das erste Mal von IBM in der Version SMB 1.0 vorgestellt und ist außerdem unter der Bezeichnung CIFS (Common Internet File System) bekannt. Microsoft integrierte das Protokoll zur damaligen Zeit in ihren LAN-Manager[1] [8]. Zur heutigen Zeit befindet sich das Protokoll bereits in der dritten Version (3.x.x). Da für diese Sicherheitslücke ausschließlich die SMB Version 1.0 (nachfolgend SMBv1) genutzt wurde, wird in dieser Arbeit nicht näher auf die Funktionsweise der höheren Versionen eingegangen.
SMB wird meistens als ein Protokoll der Anwendungsschicht (7) oder auch der Darstellungsschicht (6) (siehe Abbildung 2.3) eingesetzt [23]. Wie Abbildung 2.3 außerdem zu entnehmen ist, kann es dabei auf unterschiedlichen darunterliegenden Protokollen fungieren. Diese Grafik ist auf das Unix Umfeld bezogen (Samba), in welchem SMB per

[1] Als LAN-Manager bezeichnet man ein Netzwerkbetriebssytem von Microsoft, welches auf Os/2 basiert (Version 1.0, 1987)

OSI					TCP/IP
Application Presentation	SMB				Application
Session	NetBIOS		NetBIOS	NetBIOS	
Transport	IPX[1]	NetBEUI	DECnet	TCP&UDP	TCP/UDP
Network				IP	IP
Link	802.2, 802.3,802.5	802.2 802.3,802.5	Ethernet V2	Ethernet V2	Ethernet or others
Physical					

Abbildung 2.3: SMB im OSI-Schichtenmodell [30]

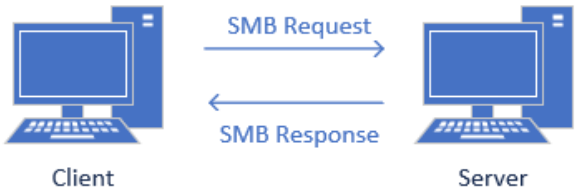
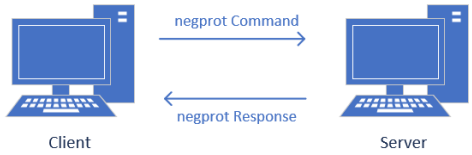


Abbildung 2.4: SMB Request-Response Modell

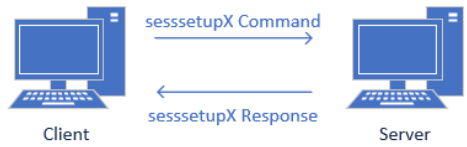
NetBIOS über TCP/IP betrieben wird (Port 139). Im Windows Umfeld kann dies auch genutzt werden, jedoch ist es hier außerdem möglich, SMB direkt über TCP/IP (ohne NetBIOS) zu verwenden (Port 445).
SMB ist ein Client-Server Protokoll, welches mit dem Request-Response Verfahren arbeitet [30]. Das bedeutet, dass auf jede Nachricht des Clients im Regelfall immer eine Antwort des Servers folgt (siehe Abbildung 2.4). Eine Ausnahme hiervon sind allerdings Transaktionen. Diese werden im Verlauf dieses Abschnitts näher beschrieben.
Ein Verbindungsaufbau via SMB sieht folgendermaßen aus: (*Hinweis:* Die folgenden Abbildungen sowie 2.4 sind in Anlehnung an [30] entstanden)

1. Protokollaushandlung



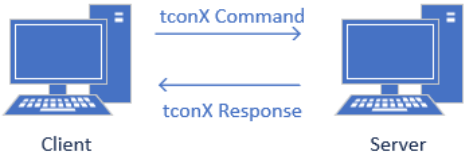
Nachdem sich der Client über ein darunterliegendes Protokoll mit dem Server verbunden hat, sendet er zunächst ein negprot Kommando, in dem er auflistet, welche SMB-Protokollvarianten er beherrscht. Wenn der Server einen dieser Dialekte akzeptiert, sendet er dem Client den Index dieses Protokolls, welches anschließend für die weitere Kommunikation genutzt wird [31].

2. Sitzungsaufbau



Nachdem das Protokoll festgelegt wurde, kann der Client sich nun beim Server einloggen. Dies geschieht mit Hilfe eines sesssetupX Kommandos, welches den Benutzernamen und das Passwort enthält. Wie ein solches Kommando aufgebaut ist, wird unter anderem in Abschnitt 2.1.3 (Bug C) beschrieben. In der Antwort teilt der Server mit, ob die Authentifizierung erfolgreich war und übermittelt dem Client bei Erfolg zusätzlich eine UID (UserID), die mit allen zukünftigen SMB's ab sofort mitgeschickt werden muss (ein "SMB" soll in dieser Arbeit eine einzelne SMB-Nachricht bezeichnen) [30].

3. Verbindung auf Laufwerk



Um sich nun mit einem Laufwerk verbinden zu können, schickt der Client dem Server ein tconX (Tree Connect) SMB, in welchem der Name des Laufwerks angegeben ist, auf das er sich verbinden möchte. Wenn die Verbindung zustande gekommen ist, antwortet der Server mit einer TID (TreeID - Identifikation des Laufwerks), die der Client zusätzlich zur UID, bei jedem zukünftigen SMB mitangeben muss [30].

SMB Nachricht Eine SMB Nachricht besteht aus insgesamt drei Teilen:

- Header (feste Länge)
- Parameter Block (variable Länge)
- Data Block (variable Länge)

Der Header gibt dabei an, dass es sich bei der Nachricht um eine SMB Nachricht handelt. Außerdem wird in ihm das jeweilige Kommando spezifiziert, welches ausgeführt werden soll. Bei einer Antwort vom Server enthält der Header zudem Informationen darüber, ob das Kommando ausgeführt werden konnte oder nicht. Zur richtigen Zuordnung der Nachricht sind in ihm die Felder UID, TID, PID (ProzessID - Identifikation des zuständigen Prozesses, wird vom Server vergeben) und MID (MultiplexID - Wird vom Server vergeben[2]) vorhanden [4] [Abschnitt 2.2.3].
Der Parameter und der Data Block sind variabel und hängen vom jeweiligen Kommando, welches im Header spezifiziert wurde, ab [4] [Abschnitt 2.2.3]. In Abbildung 2.5 ist der allgemeine Aufbau einer SMB Nachricht dargestellt, inklusive der Beschreibung wichtiger Inhalte.
Laut Microsoft sind SMB Nachrichten in dieser Art und Weise strukturiert, da das Protokoll ursprünglich als “Remote Procedure Call System“ konzipiert wurde. Die Parameter Werte sollten dabei die Werte sein, die einer Funktion übergeben werden und der Data Block sollte die Daten enthalten, die über das entsprechende Kommando beispielsweise

[2]Eine genaue Definition zu dieser ID ist in der Dokumentation von Microsoft nicht zu finden. Laut [16] wird sie zur richtigen Zuordnung des jeweiligen SMB innerhalb eines Prozesses genutzt.

geschrieben werden. Diese Differenzierung wurde trotz Weiterentwicklung des Protokolls generell beibehalten [4] [Abschnitt 2.2.3].

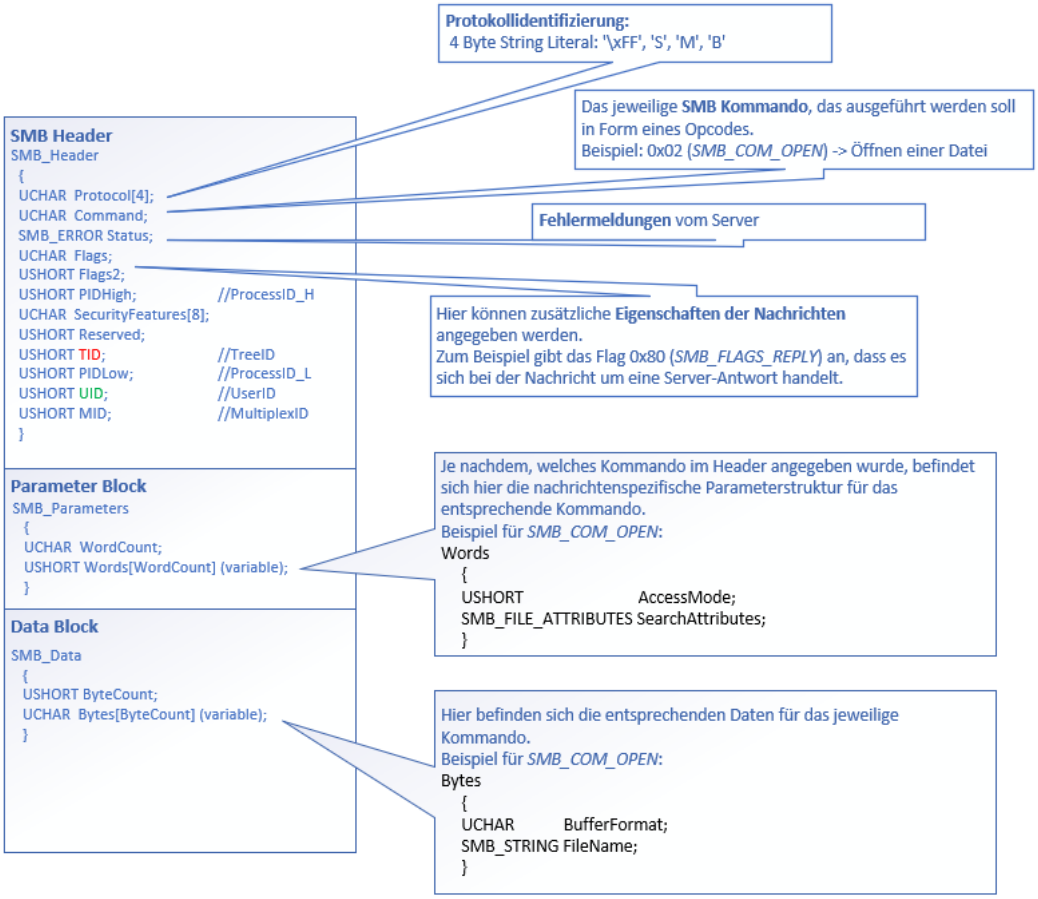


Abbildung 2.5: SMB Nachricht, auf Basis von [4] [Abschnitt 2.2.3.1-3]

Transaktionen Unter den SMB Kommandos existieren auch so genannte *Transaktionen*. Dabei handelt es sich um generische Operationen, mit dessen Einsatz man auf zusätzliche Subkommandos zugreifen kann. Diese Kommandos wiederum kann der Client nutzen, um auf dem Server auf erweiterte Funktionen zugreifen zu können [4] [Abschnitt 3.2.4.1.5]. Man kann sich Transaktionen wie "SMB in SMBs" vorstellen (Abbildung 2.6). Das bedeutet, dass Transaktionen ihre eigenen Parameter- bzw. Datenblöcke besitzen und somit in der Lage sind, ihre eigenen Subkommandos zu verwenden. Dies ist insofern wichtig zu verstehen, da eines dieser Subkommandos dazu genutzt wurde, um die Sicherheitslücke ausnutzen zu können. Von diesen Transaktionen existieren dabei drei verschiedene Arten:

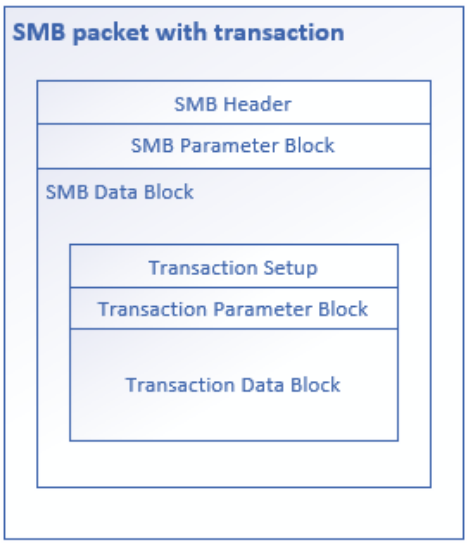


Abbildung 2.6: SMB Paket mit Transaktion, in Anlehnung an [15]

- SMB_COM_TRANSACTION *(nachfolgend vereinfacht als Trans)*
- SMB_COM_TRANSACTION2 *(nachfolgend vereinfacht als Trans2)*
- SMB_COM_NT_TRANSACT *(nachfolgend vereinfacht als NT_Trans)*

Hierbei handelt es sich um **primäre** Transaktionen. Falls die Datenmenge einer Primärtransaktion zu groß für eine SMB Nachricht werden sollte, bietet SMB die Möglichkeit, die Daten auf mehrere Transaktionen aufzuteilen. Sollte das der Fall sein, dann kommen die so gennanten sekundären Transaktionen zum Einsatz. Jedem Primärtransaktionstyp ist dabei eine Sekundärtransaktion zugeordnet:

- Trans → Trans_Secondary
- Trans2 → Trans2_Secondary
- NT_Trans → NT_Trans_Secondary

Der Ablauf eines solchen Prozesses lässt sich folgendermaßen darstellen:

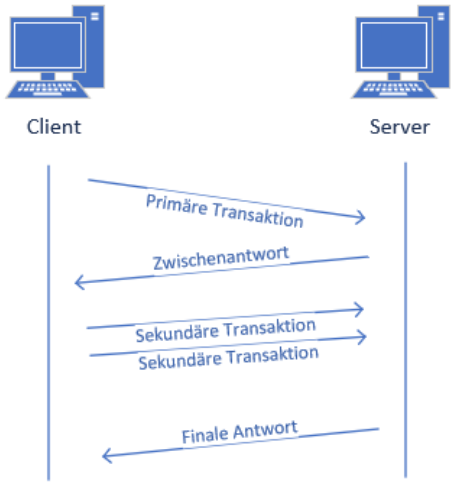


Abbildung 2.7: SMB Transaktionen, in Anlehnung an [4] [Abschnitt 3.2.4.1.5]

Zu Beginn schickt der Client dem Server eine SMB Nachricht, welche die jeweilige Primärtransaktion enthält. Diese Anfrage enthält im "Transaction Parameter Block" (Abbildung 2.6) unter anderem die zwei Parameter: **DataCount** und **TotalDataCount** (Abbildung 2.8. **DataCount** beinhaltet dabei die Größe der Transaktionsdaten (Datenblock) in der aktuellen SMB Nachricht und **TotalDataCount** die Größe der Transaktionsdaten, welche alle Datenblöcke der Transaktion umfasst. Sollte **DataCount** also kleiner sein als **TotalDataCount**, so weiß der Server, dass weitere Daten erwartet werden und es muss mindestens eine Sekundärtransaktion auf die Primärtransaktion folgen [4] [Abschnitt 2.2.4.62]. In dem Schema ist zu erkennen, dass der Server nur einmal, nach der initialen Primärtranskation, eine Antwort liefert (Status der Anfrage) und der Client daraufhin seine weiteren sekundären Transaktionen, ohne Antwort vom Server, hinterherschicken kann (Performanz).

2.1.3 Analyse

Nachdem ein kurzer Überblick über die Funktionsweise und Eigenschaften von SMB gegeben wurde, sind die Grundlagen geschaffen, um sich mit der Analyse der Sicherheitslücke zu beschäftigen. Bei EternalBlue handelt es sich dabei um ein Zusammenspiel von mehreren Bugs, welche in den nachfolgenden Abschnitten jeweils einzeln analysiert werden. Dabei ist zu beachten, dass Bug B erst durch Bug A ermöglicht wird. Bug C ist vollkommen unabhängig zu betrachten. Die Zusammenhänge der drei Bugs werden im darauffolgenden Abschnitt (2.1.4 Exploitation Flow) deutlich gemacht. Um beim Lesen

bereits ein besseres Verständnis vom Sachverhalt zu bekommen, wird zu Beginn des jeweiligen Bugs bereits das Resultat kurz erläutert.
Hinweis: Die Codeausschnitte aus dem Windows Kernel (Reverse Engineering Ergebnisse), welche in diesem Abschnitt gezeigt werden, wurden nicht selbstständig angefertigt, sondern den jeweiligen Quellen entnommen und für diese Arbeit etwas modifiziert.

Bug A: Falsche Transaktionsverarbeitung

Resultat: Mit Hilfe von diesem Bug lässt sich in einem Transaktionstypen eine größere Datenmenge übertragen, als ursprünglich dafür vorgesehen.

Wie in Abschnitt 2.1.2 beschrieben, bietet SMB die Möglichkeit, Transaktionen zu nutzen. Die unterschiedlichen Arten der Transaktionen unterscheiden sich unter anderem auch durch die Größe des Parameters **TotalDataCount** (Abbildung 2.8). Das bedeutet,

```
SMB_COM_TRANSACTION2
SMB_Parameters
{
  UCHAR  WordCount;
  Words
   {
    USHORT TotalParameterCount;
    USHORT TotalDataCount;     // (2 Bytes)
    USHORT MaxParameterCount;
    USHORT MaxDataCount;
    UCHAR  MaxSetupCount;
    UCHAR  Reserved1;
    USHORT Flags;
    USHORT DataCount;
     ......
   }
}
```

```
SMB_COM_NT_TRANSACT
SMB_Parameters
{
  UCHAR  WordCount;
  Words
   {
    UCHAR  MaxSetupCount;
    USHORT Reserved1;
    ULONG  TotalParameterCount;
    ULONG  TotalDataCount;     // (4 Bytes)
    ULONG  MaxParameterCount;
    ULONG  MaxDataCount;
    USHORT DataCount;
     ......
   }
}
```

Abbildung 2.8: Trans2 vs. NT_Trans [4] [Abschnitt 2.2.4.46.1 u. 2.2.4.62.1]

dass über eine Transaktion vom Typ *NT_Trans* eine größere Menge an Daten (ULONG = 4 Bytes) als beim Typ *Trans2* (USHORT = 2 Bytes) übermittelt werden können.
Im Fall von EternalBlue wird eine Primärtransaktion vom Typ *NT_Trans* mit **TotalDataCount** > 0xffff gesendet. Im Regelfall sollten von nun an alle *_Secondary* Transaktionen vom Typ *NT_Trans_Secondary* sein, da die Transaktion mit diesem Typ eingeleitet wurde. Dies wird vom Server allerdings nicht überprüft. Man ist in der Lage die nachfolgenden, sekundären Transaktionen eines beliebigen Typs zu senden. EternalBlue nutzt in diesem Fall dazu *Trans2_Secondaries*.
Es ist egal, von welchem Typ die sekundären Transaktionen sind. Es muss nur sichergestellt sein, dass diese die gleiche UserID, TreeID, ProcessID und MultiplexID, wie die Primärtransaktion, beinhalten. So können sie serverseitig der gleichen Transaktion zugeordnet werden (Abbildung 2.9) [4] [Abschnitt 2.2.4.47.1].

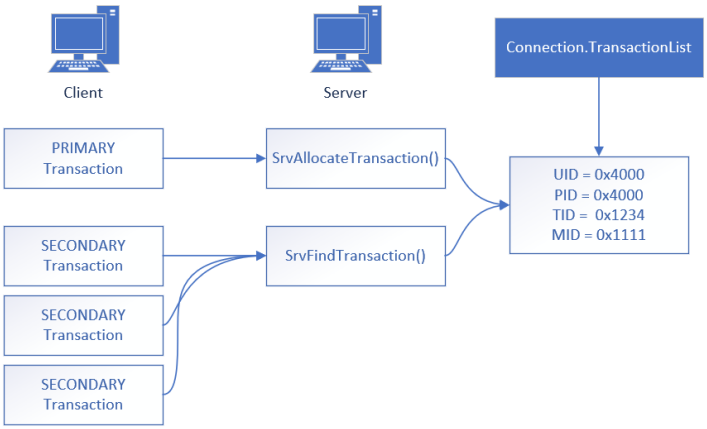


Abbildung 2.9: Transaktionsverarbeitung, in Anlehnung an [15]

Sobald alle sekundären Transaktionen eingetroffen sind, beginnt der Server mit der Verarbeitung der gesamten Transaktion [4] [Abschnitt 3.2.4.1.5]. Dabei gibt es keinerlei Validierung, welcher Transaktionstyp den Prozess ursprünglich gestartet hat [21]. Der Server ruft für die Verarbeitung die Funktion für den jeweiligen letzten sekundären Transaktionstypen auf. Im Exploitfall wird die Transaktion also als *NT_Trans* gestartet, aber im Nachhinein vom Server als *Trans2* weiterverarbeitet. Das bedeutet, dass durch diesen Bug Daten größer als 0xffff Bytes gesendet werden können, obwohl dies für eine *Trans2* nicht vorgesehen ist.

Bug B: Fehlerhafte Datentypkonvertierung

Resultat: Hierbei handelt es sich um den Bug, der durch eine falsche Datentypkonvertierung, den Buffer Overflow ermöglicht. Ausschlaggebend dafür ist die große Datenmenge, die durch Bug A ermöglicht wurde zu senden.

SMB bietet die Möglichkeit, beim Erstellen einer Datei, die so genannten “Extended Attributes“ von einer anderen Dateistruktur in die entsprechende NT-Struktur zu übertragen. Im Fall von EternalBlue wird dabei die Umwandlung einer OS/2-Struktur in die NT-Struktur genutzt. Diese “Extended Attributes“ einer Datei kann man sich wie eine Liste mit Schlüssel-Wert-Paaren vorstellen (Abbildung 2.10). Man bezeichnet diese im SMB Umfeld auch als “Full Extended Attributes“, falls sie einen Schlüssel und einen Wert besitzen [16]. In dieser Arbeit wird ein einzelnes Schlüssel-Wert-Paar abgekürzt als ein “FEA“ bezeichnet. Diese Art von Attributen zählt dabei nicht zu den herkömmlichen Dateiattributen wie Erstellungszeit, Berechtigungen etc., sondern dient zur Erweiterung

```
jboe@kali:~/Desktop/Bachelorarbeit/Eternalblue/Extended File Attributes$ getfattr -d TestDatei.txt
# file: TestDatei.txt
user.author="Johannes Boermann"
user.comment="Test f. Extended File Atrributes"
```

Abbildung 2.10: Full Extended Attributes in Linux

dieser. Sie werden beispielsweise dazu genutzt, um einem Dateisystem zusätzliche Funktionalität zu geben, z.B. zur Umsetzung von ACLs (Access Control Lists) [3]. Außerdem wurden sie laut dem IT-Sicherheitsforscher Seon Dillon für die Implementierung des WSL (Windows Subsystem for Linux) genutzt, um beispielsweise mit Case-Sensitive Dateinamen unter Windows umgehen zu können [16].
Um beim Erstellen einer Datei über SMB diese FEAs mitangeben zu können, existiert dafür das Subkommando *TRANS2_OPEN2* der Transaktion *Trans2* (Abbildung 2.11). Dieses Subkommando enthält im Datenblock eine Liste der FEAs, die der Datei zu-

```
TRANS2_OPEN2
Trans2_Data
  {
  SMB_FEA_LIST ExtendedAttributeList;
  }
```

Abbildung 2.11: Trans2_Open2 Subkommando

geordnet werden soll. Wenn diese FEAs im alten OS/2 (SMB_FEA) Format gesendet werden, bietet Microsoft die Möglichkeit, diese in das NT Format (FILE_FULL_EA_INFORMATION) umzuwandeln (Abbildung 2.12). *Hinweis:* Für

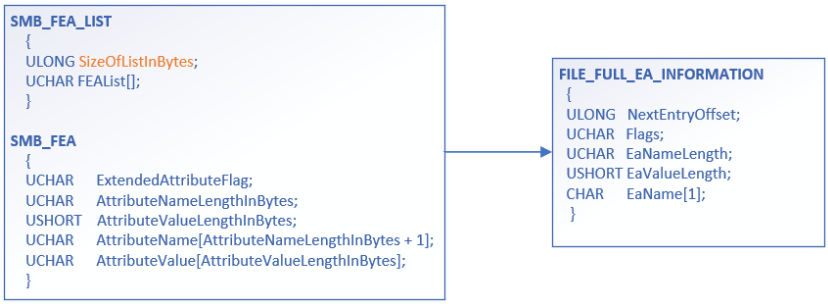


Abbildung 2.12: Os/2 FEAs zu NT FEAs

die NT FEAs existiert keine eigene Listenstruktur, sondern man iteriert über die einzelnen Einträge bis NextEntryOffset == 0 (Abbildung 2.12). Trotzdem werden im weiteren Verlauf der Arbeit die zusammengehörenden NT FEAs zur Vereinfachung als NT FEA-Liste bezeichnet.

Durch Bug A hat man nun die Möglichkeit eine SMB_FEA_LIST größer als 0xffff Bytes zu senden, was über eine "reguläre" *Trans2* Transaktion normalerweise nicht möglich ist. Im Fall von EternalBlue wird als SMB_FEA_LIST.SizeOfListInBytes = 0x10000 gesetzt.
Die SMBv1 Implementierung befindet sich im Treiber **srv.sys** von Windows. Wenn vom Client das Subkommando *TRANS2_OPEN2* genutzt wird, ruft **SrvSmbOpen2()** auf der Serverseite die Funktion **SrvOs2FeaListToNt()** auf, welche dazu dient eine Os/2 FEA-Liste in die entsprechende NT Struktur umzuwandeln. Diese wiederum ruft die Funktion **SrvOs2FeaListSizeToNt()** auf, die berechnet, wie viel Speicher für die "neue" NT FEA-Liste reserviert werden muss. Jedes NT-FEA ist etwas größer als ein Os/2-FEA, weshalb die NT FEA-Liste im Normalfall immer größer sein muss als die Os/2 FEA-Liste.
Wie man Punkt 3 in Abbildung 2.13 entnehmen kann, befindet sich der kritische Teil in der Funktion **SrvOs2FeaListSizeToNt()**. Diese dient nicht nur dazu, die benötige Größe für die NT FEA-Liste zu berechnen, sondern korrigiert gleichzeitig die Größe der erhaltenen Os/2 FEA-Liste (SizeOfListInBytes) um das "letzte" zu große FEA (FEA Record), falls dieses über die Größe der vom Client gelieferten Os/2 FEA-Liste (in diesem Fall 0x10000) hinausgehen sollte. In anderen Worten: Man will an dieser Stelle sicherstellen, dass der Client keine Listeneinträge schickt, welche summiert über die von ihm angegebene Listengröße hinausgehen.
In der kritischen Codezeile (Abbildung 2.14) ist zu sehen, dass für die Berechnung der neuen Listengröße der Zeiger auf das Ende der Liste (ohne den zu großen, letzten Eintrag) vom Zeiger auf den Anfang der Liste subtrahiert wird und anschließend die neue Größe in SMB_FEA_LIST(Os/2 Liste)->SizeOfListInBytes geschrieben wird.
Das Problem: Der Wert SizeOfListInBytes wurde in SMB_FEA_LIST (siehe Abbildung 2.12) als ULONG (DWORD = 32 Bit) definiert und wird für diese Anpassung in ein WORD[3] (16 Bit) konvertiert. Das hat zur Folge, dass bei SizeOfListInBytes nur die unteren 16 Bit (LOWORD) verändert werden und die oberen 16 Bit (HIGHWORD) bestehen bleiben (siehe Abbildung 2.15).

Das bedeutet, dass sich SizeOfListInBytes dadurch nicht, wie eigentlich vorgesehen, verkleinert, sondern **vergrößert**. SizeOfListInBytes der Os/2 FEA-Liste beträgt nach der fehlerhaften Umwandlung 0x1ff5d, wohingegen die NT FEA-Liste nur eine Größe von

[3]Ein von Microsoft definierter Datentyp, welcher einem "16 Bit unsigned Integer" entspricht, siehe: https://docs.microsoft.com/en-us/windows/desktop/winprog/windows-data-types

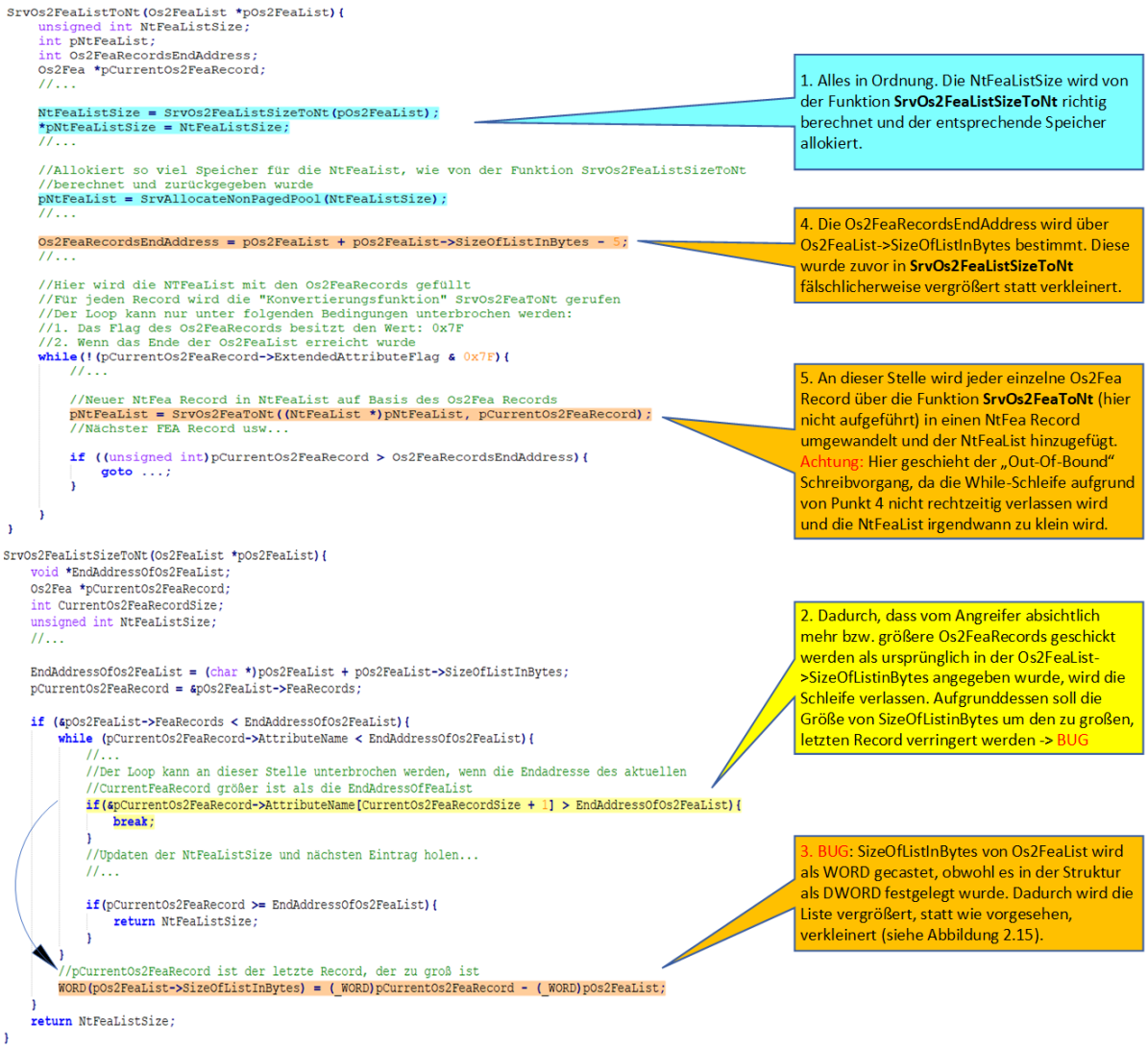


Abbildung 2.13: srv.sys Funktionen zur Umwandlung der Os/2 FEA-Liste in das NT Format, in Anlehnung an [21]

```
WORD(pOs2FeaList->SizeOfListInBytes) = (_WORD)pCurrentOs2FeaRecord - (_WORD)pOs2FeaList;
```

Abbildung 2.14: Kritische Codezeile [vgl. Abbildung 2.13]

Ziel: Anpassung Os2FeaList->SizeOfListInBytes von 0x1000 zu 0x55fd		
	DWORD SizeOfListInBytes	
	Hex	Binär
Ursprünglicher Wert	0x10000	00000000000000001000000000000000 (HIWORD / LOWORD)
Nach (fehlerhafter) Konvertierung	0x155fd	00000000000000010101010111111101

Abbildung 2.15: Fehlerhafte Datentypkonvertierung

0x10fe8 Bytes besitzt (diese Werte stammen aus den Analyseergebnissen von [25]). Im Normalfall sollte es aber der Fall sein, dass die NT FEA-Liste immer größer ist als die Os/2 FEA-Liste, da, wie schon erläutert, ein NT FEA-Record größer ist als ein Os/2 FEA-Record. Aufgrund dieses Fehlers wird in der Funktion **SrvOs2FeaListToNt()** die Os2FeaRecordsEndAddress falsch bestimmt (Punkt 4, Abbildung 2.13), welche unter anderem die Abbruchbedingung bei der Konvertierung der einzelnen Records beeinflusst. Das bedeutet wiederum, dass bei der Umwandlung (Punkt 5, 2.13) die NT FEA-Liste irgendwann zu klein wird und folglich über den bereitgestellten Speicherbereich der Liste hinausgeschrieben werden kann.
Um dies ausnutzen zu können, werden von EternalBlue zuerst 605 "leere"[4] FEAs geschickt, anschließend ein etwas größeres FEA und darauffolgend das so genannte "Buffer Overflow FEA" (Abbildung 2.16). Mit Hilfe von diesem wird zum einen die Größe der Os/2 FEA-Liste (0x10000) letztendlich überschritten, so dass der Kontrollfluss in die entsprechende kritische Codezeile gerät (Abbildung 2.14). Zum anderen wird auch damit die Größe des reservierten Speichers für die NT FEA-Liste (0x10fe8) überschritten, was letztendlich zum Speicherüberlauf führt. Das anschließend gesendete "Invalid FEA"[5] (Abbildung 2.16) dient dem Angreifer dazu, einen Hinweis darauf zu erhalten, ob der Speicherüberlauf erfolgreich war. Der Gedanke dahinter ist etwas unklar. Man vermutet folgendes: Wenn der Server nach dem "Buffer Overflow FEA" noch ein ungültiges FEA abarbeitet und dem Angreifer eine entsprechende Meldung darüber zurückliefert (*STATUS_INVALID_PARAMETER*), lässt sich darauf schließen, dass der Windows Kernel

[4] Leere bzw. NULL-FEAs bezeichnen hier FEAs, welche einen leeren Attributnamen und Attributwert besitzen
[5] Hierbei handelt es sich um ein FEA, welches als ExtendedAttributeFlag (siehe Abbildung 2.15) einen nicht definierten Wert enthält

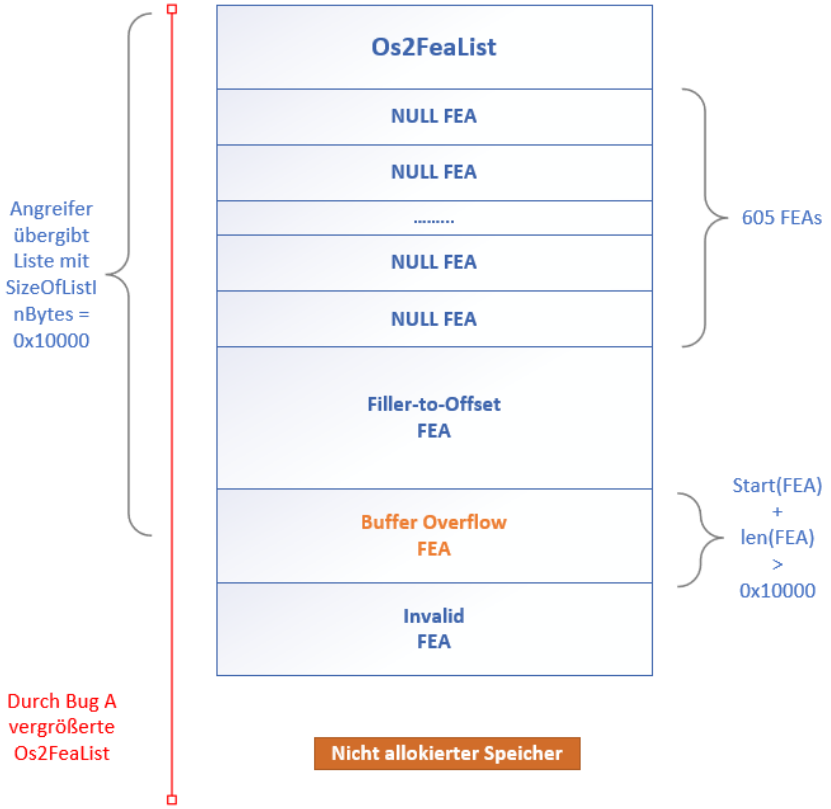


Abbildung 2.16: Aufbau der schädlichen Os/2 FEA-Liste, in Anlehnung an [15]

bei diesen Aktivitäten zumindest nicht abgestürzt ist [16].
Ungeklärt bleibt in diesem Fall jedoch, warum Microsoft die FEAs, welche zu groß für die Liste sind, nicht einfach abweist.
Welcher Bereich im Speicher durch diesen Prozess genau überschrieben wird, folgt in Abschnitt 2.17 (Exploitation Flow).

Bug C: Große Kernel Pool Allokation

Resultat: Dieser Bug ist unabhängig zu betrachten und dient zur "Vorbereitung" des Speicherbereichs, so dass der Buffer Overflow in die korrekte Stelle im Kernel Speicher trifft.

Wie in Abschnitt 2.1.2 bereits erläutert, muss sich der Client immer erst beim Server authentifizieren um weitere Aktionen per SMB ausführen zu können. Dafür ist es zwingend notwendig, dass der Client eine so genannte *SMB_COM_SESSION_SETUP_ANDX* Anfrage schickt [4] [Abschnitt 2.2.4.53]. Diese Nachricht existiert in zwei unterschiedli-

chen Formaten:

1. LM und NTLM Authentifizierung: Authentifizierungsverfahren, welches für den LAN Manager (LM) und NT LAN Manager (NTLM) genutzt wird.

2. NTLMv2: Weiterentwicklung von NTLM, welche mit Windows NT 4.0 SP4 eingeführt wurde.

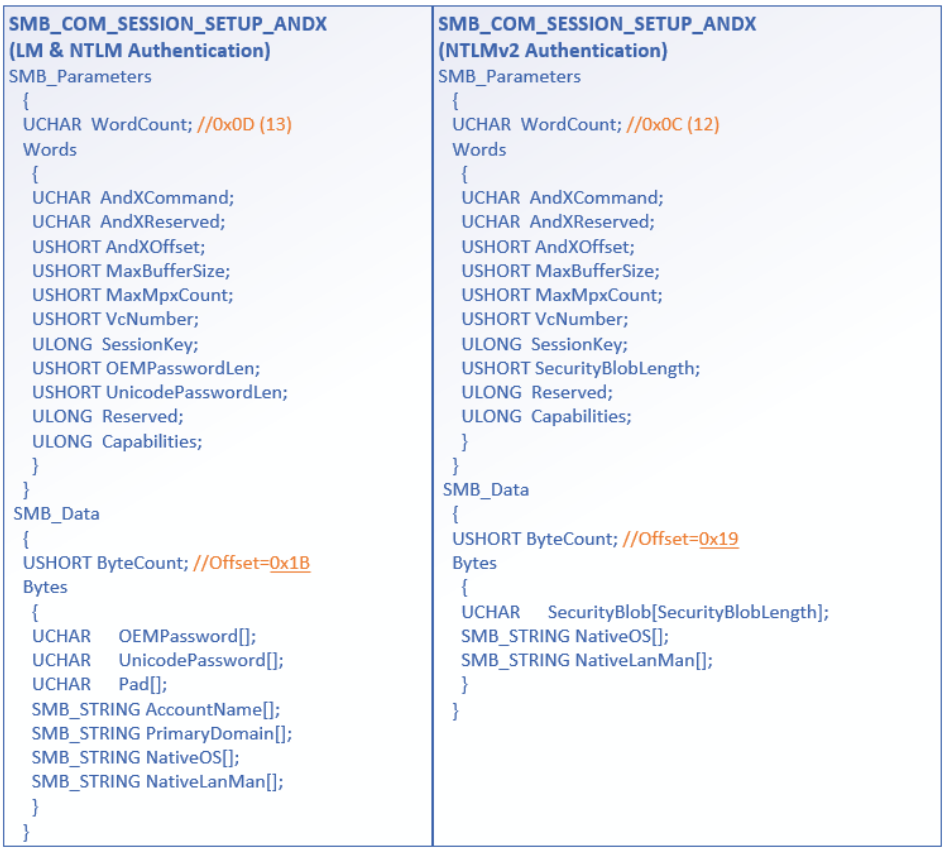


Abbildung 2.17: LM/NTLM vs. NTLMv2 [4] [Abschnitt 2.2.4.53.1 u. 2.2.4.6.1]

Diese beiden Formate unterscheiden sich durch ihren WordCount (LM/NTLM = 13 u. NTLMv2 = 12). Dies hat zu Folge, dass sich der ByteCount im Datenblock jeweils an einem anderen Offset befindet (siehe Abbildung 2.17). Als Angreifer hat man die Möglichkeit, eine NTLMv2 Anfrage zu senden, die aber fälschlicherweise als LM/NTLM Anfrage interpretiert wird. Im Code (Abbildung 2.18) ist zu erkennen, dass wenn man eine Session-Anfrage vom NTLMv2 Typ schickt, mit dem Parameter Capabilities = CAP_EXTENDED_SECURITY aber **ohne** das SMB-Header Flag Flags2 = FLAGS2_EXTENDED_SECURITY, der else Block abgearbeitet und die Anfrage als LM/NTLM interpretiert wird. Das hat zur Folge, dass die Funktion GetNtSecurityParameters() nun an dem falschen Offset (0x1B) den ByteCount ausliest [21]. Zwar findet in

```
BlockingSessionSetupAndX(request, smbHeader)
{
    //.....

    // Word Count überprüfen
    if(! (request->WordCount == 13 || (request->WordCount == 12 && (request->Capabilities & CAP_EXTENDED_SECURITY)) ){
        // Fehler und return...
    }

    //....

    if((request->Capabilities & CAP_EXTENDED_SECURITY) && (smbHeader->Flags2 & FLAGS2_EXTENDED_SECURITY)){
        //Hierbei handelt es sich um eine NTLMv2 Anfrage (Extended Security)
        GetExtendSecurityParameters(request); // Parameter extrahieren und Speicher allokieren
        SrvValidateSecurityBuffer(request); // Authentifizierung durchführen
    }
    else{
        //LM & NTLM Anfrage
        GetNtSecurityParameters(request); // Parameter extrahieren und Speicher allokieren
        SrvValidateUser(request); // Authentifizierung durchführen
    }
    //.....
}
```

Abbildung 2.18: SessionSetupAndX Funktion, in Anlehnung an [21]

der Funktion SrvValidateSmb() eine Validierung des ByteCounts statt, bevor die SMB Anfrage an den “Request Handler“ weitergreicht wird [34], jedoch ist das zwecklos, wenn im weiteren Verlauf der Wert an einer anderen Stelle ausgelesen wird. Man ist dadurch also in der Lage eine sehr kleine NTLMv2 Anfrage zu senden (ByteCount = 0x16), für welche aber im Nachhinein, durch den falsch ausgelesenen Wert, viel mehr Speicher allokiert werden kann (bis zu 0x20000 Bytes) [34]. In anderen Worten: Der Angreifer hat hierdurch die Möglichkeit, variablen und sehr viel Speicher im non-paged Kernel Pool, über sehr kleine Anfragen, allokieren zu lassen [21].
Diese Technik wird von dem Exploit insgesamt zwei Mal verwendet. Beim ersten Mal handelt es sich um eine Allokation der Größe 0x10000. Sie wird dazu genutzt, um eine so genannte “Pre Hole Connection“ aufzubauen. Bei der zweiten Verwendung handelt es sich um eine Allokation der Größe 0x11000, welche das “Hole“ (Speicherloch) darstellt, in das, nachdem die Verbindung wieder geschlossen wurde, die NT FEA-Liste (0x10fe8) geschrieben werden soll [25]. Darauf wird im nachfolgenden Abschnitt näher eingegangen.
Zusammenfassend zu diesem Bug lässt sich also sagen, dass hiermit eine Möglichkeit geschaffen wurde, um Speicher, welcher weitaus Größer ist als die eigentliche SMB-Nachricht, im non-paged Kernel Pool allokieren zu lassen. Ein weiterer Vorteil ist, dass man diesen Speicherbereich, durch das Schließen der Verbindung, gezielt wieder freigeben lassen kann. Abgesehen davon, handelt es sich bei diesem Bug um einen eher als unkritisch einzustufenden Fehler. Außer für die soeben genannten Punkte, lässt sich dieser Bug nicht weiter einsetzen.

2.1.4 Exploitation Flow

Nachdem die drei Bugs näher erläutert wurden, wird in diesem Kapitel der Ablauf des Exploits betrachtet.
Es sei angemerkt, dass für den allgemeinen SMB-Verbindungsaufbau bei jeder einzelnen Aktion von einem anonymen Login auf das so genannte IPC$-Share von Windows Gebrauch gemacht wird (null-Sitzung)[6]. Diese Freigabe ist auf Windows Rechnern standardmäßig vorhanden und kann auch nur unter einigen Umständen deaktiviert werden. Zum besseren Verständnis wird in dieser Darstellung nicht auf SMB Details eingegangen, sondern Schritt für Schritt gezeigt, was im Kernel Speicher passiert.
Hinweis: Die folgende grafische Darstellung wurde von [15] übernommen. Einige Teile der Beschreibungen beruhen auf den zugehörigen Aussagen [16].

1. Aufbau des Kernel Pools vor dem Exploit

Zu diesem Zeitpunkt befinden sich zufällige Dinge im Speicher. Außerdem besitzt SrvNet

[6] Für weitere Informationen zur IPC$-Freigabe und Nullsitzungsverhalten siehe: https://support.microsoft.com/de-de/help/3034016/ipc-share-and-null-session-behavior-in-windows, zuletzt abgerufen am: 23.03.2019

im Regelfall etwas "Tanslation Lookaside Buffer"[7]. SrvNet.sys ist der Treiber, welcher die entsprechende Funktionalität für **SMBv2** beinhaltet. Weshalb SMBv2 in diesem Fall von Bedeutung ist, folgt in den nächsten Schritten.

2. Senden der Os/2 FEA-Liste ohne die letzte Nachricht

Es wird die gesamte Os/2 FEA-Liste gesendet, bis auf die letzte *Trans2_Secondary* Nachricht, welche das "Buffer Overflow FEA" (siehe Abbildung 2.16) enthält. Da die Transaktion also noch nicht vollständig beim Server eingetroffen ist, bedeutet das, dass noch kein Speicher für die neue NT Fea-Liste reserviert wird und mit der Abarbeitung noch nicht begonnen wurde.

3. Senden einer beliebigen Anzahl an SMBv2 Nachrichten

In diesem Schritt sendet man eine beliebige Anzahl an SMBv2 Nachrichten, die dazu führen, dass der vorhandene "lookaside Buffer" von SrvNet aufgebraucht wird. Dies hat zur Auswirkung, dass von nun an, bei jeder SMBv2 Anfrage, neuer Speicher im Pool allokiert wird. *Hinweis:* Man nutzt SMBv2 Nachrichten, da diese von SrvNet.sys verarbeitet werden. Dies ist wichtig, da man im späteren Verlauf in den Speicher von SrvNet überlaufen möchte.

4. Senden der Session "pre hole" Buffer Nachricht (Bug C Tei1)

Der Teil 1 von Bug C wird gesendet, welcher Speicher in einer etwas kleineren Größe (0x10000), als die NT FEA-Liste (0x10fe8) allokiert.

5. Senden der Session "hole" Buffer Nachricht (Bug C Teil2)

[7]Ein Translation Lookaside Buffer (TLB) ist ein Zwischenspeicher, in welchem sich (zum schnelleren Zugriff) bereits virtueller Speicher befindet, der schon vorsorglich physikalischen Speicheradressen zugeordnet wurde

An dieser Stelle wird der Teil 2 von Bug C gesendet, welcher die richtige Größe an Speicher allokiert (0x11000), in welchen die NT FEA-Liste passt.

6. Schließen des Session “pre hole“ Buffers (Bug C Teil1)

In diesem freigewordenen Speicherbereich sollen alle Allokationen stattfinden, die unabhängig von dem Exploit im System stattfinden. Es dient außerdem dazu, dass in der Zeit, während das eigentliche Speicherloch freigemacht wird, keine anderen Dinge in diesem Loch allokiert werden, sondern stattdessen gezielt in diesen Bereich wandern. Man kann also sagen, dass es als Schutzvariante des Exploits dient, welche die Chancen auf erfolgreiche Ausführung erhöhen soll.

7. Senden einer weiteren beliebigen Anzahl an SMBv2 Nachrichten

Weitere SMBv2 Nachrichten werden gesendet, mit dem Ziel, dass eine Nachricht dafür sorgt, dass SrvNet Speicher nach dem platzierten Loch allokiert wird, in welchen übergelaufen werden soll. Diese Technik wird auch als “Heap Spraying“ bzw. in diesem Zusammenhang als “Kernel Grooming“ bezeichnet. Sollte bei den ersten Versuchen der Überlauf nicht den Speicherbereich von SrvNet getroffen haben, so erhöht EternalBlue, in seiner Originalfassung, selbständig bei den nächsten Versuchen die Anzahl der SMBv2 Nachrichten, was die Chancen dafür erhöht.

8. Freigeben des Lochspeichers

Der “hole Buffer“ wird freigegeben und der Speicherbereich wird zum eigentlichen Speicherloch.

9. Senden des letzten Teils der Transaktion

Der letzte Teil der Transaktion wird gesendet. Das hat zur Folge, dass srv.sys nun mit der Umwandlung der FEA-Liste beginnt. Die Chancen wurden durch die geleistete Vorarbeit erhöht, dass die NT FEA-Liste in das zuvor freigemachte Speicherloch geschrieben wird. Durch die Schritte zuvor besteht außerdem eine hohe Wahrscheinlichkeit, dass sich nach der NT FEA-Liste ein Speicherbereich von SrvNet befindet, der durch das Auslösen von Bug B, nun überschrieben wird.

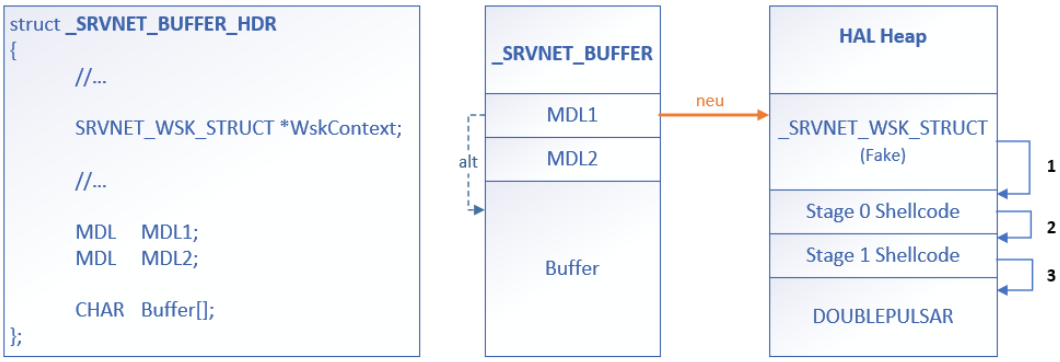


Abbildung 2.19: SrvNetHeader Überschreibung, in Anlehnung an [15]

Welche Struktur in diesem Fall genau überschrieben wird, ist in Abbildung 2.19 zu sehen. Es handelt sich hierbei um einen Header von SrvNet. Dieser enthält unter anderem zwei entscheidende Felder: Eine Memory Descriptor List (MDL)[8] und einen Zeiger auf SRVNET_WSK_STRUCT.

Hinweis: Ab diesem Zeitpunkt hängt es vom jeweiligen Betriebssystem ab, wie der weitere Ablauf des Exploits erfolgt. In dieser Arbeit wird der nachfolgende Prozess anhand von Windows 7 erläutert.

Die MDL wird so überschrieben, dass sie nicht mehr auf den für die SMBv2 Verbindung vorgesehenen Buffer zeigt, sondern stattdessen auf den Heap des Hardware Abstraction Layers (HAL)[9] [16]. Dieser hatte bis in die späteren Versionen von Windows 10 eine statische Adresse und war somit nicht über Address Space Layout Randomization (ASLR) geschützt (Beispiel Windows 7: 32 Bit: 0xffd00000, 64 Bit: 0xfffffffffd00000) [21]. Außerdem kommt hinzu, dass der Speicherbereich des HAL Heaps unter Windows 7 als ausführbar gekennzeichnet war. Dies wurde erst im Zeitalter von Windows 8/8.1

[8] "The operating system uses a memory descriptor list (MDL) to describe the physical page layout for a virtual memory buffer", siehe: https://docs.microsoft.com/en-us/windows-hardware/drivers/kernel/using-mdls, zuletzt abgerufen am: 23.03.2019

[9] dt. Hardwareabstraktionsschicht: Bezeichnet eine logische Schicht zwischen dem Betriebssystemkernel und der Hardware.

geändert.
Das bedeutet also, dass von nun an, alle Daten, die über diese Socket Verbindung per SMBv2 gesendet werden, nicht mehr in den vorgesehen Buffer geschrieben werden, sondern stattdessen direkt in den Heap des HALs. Da man nicht genau weiß, welche von den geöffneten Verbindungen (in den vorherigen Abbildungen als SrvNet "groom" Buffer, Farbe: dunkelblau gekennzeichnet) überschrieben wurde, schickt man den Shellcode an alle offenen SMBv2 Verbindungen [16].
Der erste Teil beinhaltet dabei eine gefälschte SRVNET_WSK_STRUCT, auf die man den Zeiger *WskContext im SRVNET_BUFFER_HDR zeigen lässt. Darauf folgend, wird ein zweiteiliger Shellcode gesendet und abschließend der eigentliche Backdoor, bekannt unter dem Namen "Doublepulsar".
SRVNET_WSK_STRUCT enthält unter anderem eine Funktionstabelle, welche wiederum einen Zeiger auf eine "Clean up"-Funktion enthält. Diese wird jedes Mal aufgerufen, sobald eine SMBv2 Verbindung geschlossen wird. Der Zeiger auf diese Funktion wurde in der gefälschten SRVNET_WSK_STRUCT jedoch so modifiziert, dass dieser bei Aufruf auf den ersten Teil des Shellcodes (Stage 0) zeigt. Da nun alle geöffneten SMBv2 Verbindungen geschlossen werden, führt eine Schließung davon dazu, dass es zu der Ausführung des Shellcodes im Kernel kommt [16]. An dieser Stelle hat man also den ersten Meilenstein erreicht, nämlich die "Remote Code Execution" (Abbildung 2.19, Nr. 1).
Der 'Stage 0 Shellcode' wird zu diesem Zeitpunkt unter dem Interrupt Request Level (IRQL) = **DISPATCH_LEVEL** (Abbildung 2.20, Stufe 2) ausgeführt [16]. Wenn Code in diesem Level ausgeführt wird, besitzt man keinen Zugriff auf Dinge wie "paged memory" oder viele Kernel API's, da es sich hierbei um das höchste Softwareinterruptlevel handelt. Aufgrund dessen muss sichergestellt werden, dass in der Ausführungszeit weder "page faults", noch eine zu lange Beanspruchung des Prozessors stattfinden. Zumindest ist der Zugriff auf "paged memory" allerdings nötig, um später Shellcode auch in den Usermodus einzuschleusen zu können [16]. Um dies zu ermöglichen bzw. um das IRQL auf **PASSIVE_LEVEL** zu senken, wird sich einer Technik namens "Syscall Hooking" bedient. Das bedeutet konkret, dass dafür das LSTAR-Register (Long System Target-Adress Register) der CPU überschrieben wird [16]. Dieses zählt zu den Model Specific Registers (MSR). In dieses Register wird vom Kernel beim Bootvorgang des Systems die Adresse der Funktion "nt!KiSystemCall64()" (bei 64 Bit Systemen) geschrieben. Dabei handelt es sich um die entsprechende Kernelfunktion, die bei einem Syscall aus dem Usermode, die Kontrolle im Kernel übernimmt und entsprechende weitere Funktionen zur Abarbeitung des jeweiligen Syscalls aufruft. Sie wird daher auch als Einstiegsfunk-

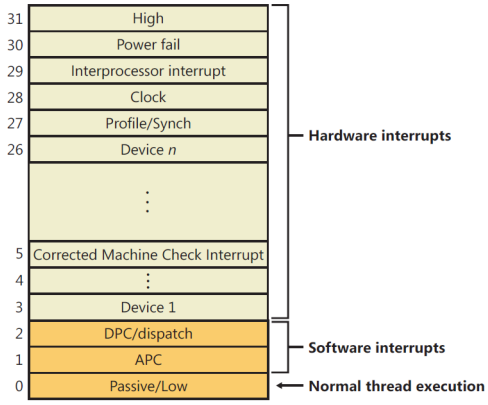


Abbildung 2.20: Windows Interrupt Request Levels (IRQLs) [29]

tion für Syscalls bezeichnet (*sysenter*). Die Adresse der Funktion wird nun ersetzt mit der Adresse zu dem Stage 1 Shellcode. Das bedeutet, dass beim nächsten Syscall, der irgendwo im System stattfindet, dieser nicht mehr abgearbeitet wird, sondern stattdessen der zweite Teil des Shellcodes aufgerufen wird. Das hat zur Folge, dass der Shellcode nun in einem Prozess Kontext aufgerufen wird (Abbildung 2.19, Nr. 2). Dies kann bei mehrkernigen System, wenn man wenig Glück hat, teils über fünf Minuten dauern. Das hat den Grund, das Syscalls möglicherweise auf den anderen Prozessoren ausgeführt werden [33]. Nach erfolgreicher Ausführung des Stage 1 Shellcodes muss allerdings sichergestellt werden, dass der "Syscall Hook" wieder entfernt wird und somit die Adresse des Registers wieder auf die ursprüngliche Funktion zeigt. Ansonsten besteht die Gefahr, dass die Windows Kernel Patch Protection (auch bekannt unter dem Namen "PatchGuard") anschlägt und sich das System daraufhin mit einem Bluescreen verabschiedet [18].
Der Stage 1 Shellcode ruft in seiner Routine abschließend den Backdoor DoublePulsar auf. Dieser hat durch die geleistete Vorarbeit nun vollen Zugriff auf Kernel API's, sowie auf "paged memory" des Usermodes (Abbildung 2.19, Nr. 3).
Natürlich kann an dieser Stelle auch anderer beliebiger Schadcode eingesetzt werden. Allerdings besitzt man mit Doublepulsar einen sehr ausgefeilten und vor allem architekturunabhängigen SMB-Backdoor [14], welcher das Nachladen von beliebiger Schadsoftware, zu jeder Zeit, auf dem infizierten Rechner ermöglicht. Dieser Backdoor wurde beispielsweise auch für die Verbreitung von WannaCry genutzt.

2.1.5 Patches

Die auf SMBv1 bezogenen Sicherheitslücken wurden von Microsoft mit dem Sicherheitsupdate MS10-17 (kritisch) am 14.03.2017 für die aktuellen Betriebssysteme geschlossen

(Ausnahmsweise etwa zwei Monate später auch für ältere Systeme, siehe Abbildung 2.2). Die folgenden CVE's wurden hiermit behoben:

- CVE-2017-0143
- CVE-2017-0144
- CVE-2017-0145
- CVE-2017-0146
- CVE-2017-0148

Wie man der Liste entnehmen kann, wurde auch EternalBlue (2017-0144) mit diesem Update behoben. Die genauen Details der Codeänderungen sind nicht einsehbar, jedoch lässt sich auf Basis der Reverse Engineering Ergebnisse von Seon Dillon [17] folgendes zu Bug A und Bug B sagen:

Bug A: Es wird ab sofort geprüft von welchem Typ eine sekundäre Transkation ist und verglichen mit dem Typ der primären Transaktion. Sollten diese nicht übereinstimmen gibt das System einen entsprechenden Fehler zurück.

Bug B: Die Datentypkonvertierung der Variable Os/2-Fealiste->SizeOfListInBytes wurde von WORD zu DWORD geändert (siehe Abbildung 2.14). Die korrekte Berechnung der neuen Listengröße wurde nun also sichergestellt.

Bug C: Zu Bug C lässt sich auf dem heutigen Kenntnisstand leider keine verlässliche Aussage treffen, inwiefern oder ob dieser behoben wurde. Laut Aussage von Worawit Wang [33] wurde er zumindest mit MS10-17 noch nicht geschlossen.

2.1.6 Fazit

Das IT-Sicherheitsunternehmen RiskSense beschreibt Eternalblue in ihrer Zusammenfassung des zugehörigen Whitepapers [18] als den *"one of the most complex exploits ever written"*.
Diese Aussage ist nach der intensiven Beschäftigung mit dieser Sicherheitslücke nachvollziehbar. Es handelt sich bei diesem Exploit um ein komplexes Zusammenspiel verschiedener Bugs, die bei richtiger Ausnutzung, zu verheerenden Folgen führen. Mit EternalBlue wurde die Möglichkeit geschaffen, beliebigen Code im Kernelmodus (ring0) eines

Windows Betriebssystems ausführen zu können. Durch die Platzierung eines passenden Backdoors wurde außerdem ermöglicht, dauerhaft eine Verbindung zu dem infizierten Rechner herstellen zu können, um beliebige Schadsoftware nachzuladen. Für dieses Szenario musste auf den entsprechenden Rechnern lediglich SMBv1 aktiviert sein und mindestens einer davon musste von außerhalb des internen Netzwerkes zugänglich sein. War dies der Fall, so konnte der Infektionszyklus, zum Teil in sehr großen Netzwerken, ins Laufen geraten, was man 2017 am Beispiel von WannaCry in der Öffentlichkeit mitverfolgen konnte. Sollten die Thesen aus Abschnitt 2.1.1 dieses Kapitels in Bezug auf die NSA stimmen, so besaßen diese viele Jahre lang ein außerordentlich mächtiges Mittel, um unentdeckt Zugriff auf verschiedenste Rechner zu erlangen.
Das nicht genug, befinden sich in dem geleakten Fuzzbunch Modul (siehe Abschnitt 2.1.1) neben EternalBlue noch ca. 15 weitere, so genannte “ready to use“ Exploits in kompiliertem Format. Jeder davon hat seinen bestimmten Einsatzzweck. Einige davon zielen auf andere fehlerhafte Implementierungen des SMBv1 Protokolls ab.
Bei SMBv1 handelt sich um ein Protokoll, dass schon vor ca. 30 Jahren entwickelt wurde und gerade in den letzten Jahren immer wieder in die Kritik geraten ist. Es ist erstaunlich, dass diese Version des Protokolls bis zum Windows Vista Zeitalter (ca. 2007, Einführung SMBv2) das standardmäßige Filesharing Protokoll im Windows Umfeld war. Erst 2014 wurde SMBv1 von Microsoft (erstmals für den Windows Server 2012 R2) als veraltet deklariert. Seit Anfang 2018 (Windows 10, Version 1709) wird SMBv1, bei der standardmäßigen Installation bestimmter Windows 10 Versionen, nicht mehr mitgeliefert.
Bei allen beschriebenen Bugs handelt es sich also um Code, der zum Teil schon in den 1980er Jahren entstanden ist. Ein Suchergebnis bei einer CVE-Datenbank[10] liefert, alleine für SMBv1, 35 CVE Einträge im Zeitraum von 2009-2017 und dabei handelt es sich “nur“ um öffentlich bekannt gewordene Sicherheitslücken. Es ist also sehr fragwürdig, warum Microsoft dieses Protokoll so viele Jahre lang unterstützt hat und sich nicht frühzeitiger strikt gegen die Verwendung eingesetzt hat, zumal 2007 der Nachfolger SMBv2 schon präsent war.
Allerdings tragen im Unternehmensbereich auch verantwortliche Systemadministratoren eine gewisse Mitschuld, wenn sie 2017 erfolgreich von der WannaCry Attacke getroffen wurden. Es wurde schon Jahre zuvor immer wieder vor der Benutzung von SMBv1 gewarnt[11]. Außerdem hat Microsoft schon vor der Verbreitung des Verschlüsselungstroja-

[10] https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=SMBv1, zuletzt abgerufen am: 24.03.2019
[11] zum Beispiel 2016 in einem offiziellen Microsoft Blog: https://blogs.technet.microsoft.com/filecab/2016/09/16/stop-using-smb1/ , zuletzt abgerufen am 28.03.2019

ners ein entsprechendes Sicherheitsupdate (MS17-010) für die aktuellen Betriebssysteme bereitgestellt. Dieses wurde entweder noch nicht aufgespielt oder es waren oft auch noch Betriebssysteme (Windwos XP etc.) im Einsatz, welche vom offiziellen Support von Microsoft, zu dieser Zeit, schon lange ausgeschlossen waren. Das solche, so genannten "Legacy Systeme", derart lange, sei es bei Unternehmen oder Privatanwendern, im Einsatz bleiben, war und ist im IT-Sicherheitsbereich schon immer ein sehr großes Problem. Diese Sicherheitslücke ist ein treffendes Beispiel dafür.
Abgeschlossen werden soll dieses Kapitel mit einem etwas ironischen Zitat des IT-Sicherheitsforschers Sean Dillon zum Thema Abschaffung von SMBv1 mit Windows 10:

"It is unclear to me why Microsoft does not deliver SMBv1 with Windows 10 anymore although it got penetration tested by the NSA for around 30 years and you can assume that it should be save now." [17]
(sinngemäß)

2.2 CVE-2018-1000115: Memcached: Insufficient Control of Network Message Volume Vulnerability

2.2.1 Hinführung

Am 28. Februar 2018 wurde der Onlinedienst GitHub (https://github.com) von der bislang stärksten veröffentlichten DDoS-Attacke getroffen. Zwischenzeitlich hatte GitHub dabei mit einem Traffic von ca. 1,35 Tbp/s in der Spitze zu kämpfen. Aufgrund des-

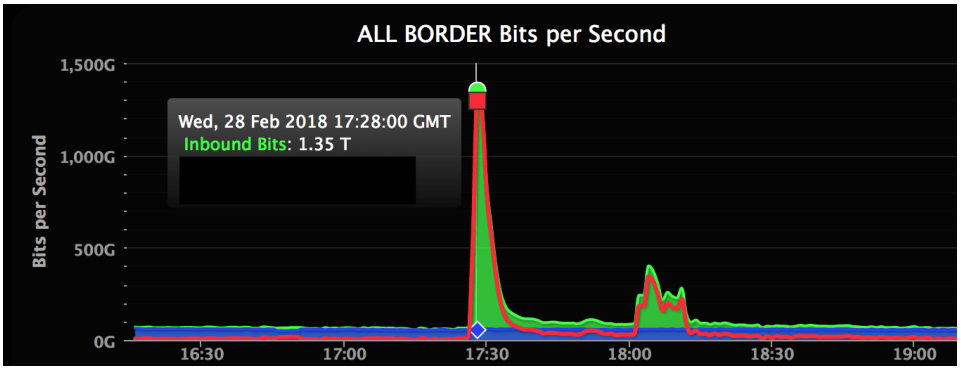


Abbildung 2.21: 1,35 Tbp/s als Spitzenwert der DDoS-Attacke [24]

sen war der Service von 17:21 bis 17:26 Uhr (UTC) nicht erreichbar und von 17:26 bis 17:30 Uhr (UTC) zwischenzeitlich nicht verfügbar [24]. Diese Ausfallzeiten erscheinen allerdings, im Vergleich zu dem Ausmaß des Angriffs, als sehr kurz. Jedoch sei an dieser Stelle erwähnt, dass es sich hierbei um einen sehr großen Dienst handelt, welcher starke Infrastrukturen besitzt und auf derartige Angriffe vorbereitet ist. Hinter GitHub steckt dabei der amerikanische Infrastruktur-Anbieter "Akamai". Dessen Vizepräsident im Bereich Web-Security Wired sagte zu dem Angriff folgendes: *"Wir haben unsere Kapazitäten so berechnet, dass wir Angriffe abwehren können, die fünfmal so stark sind, wie der bislang bekannte größte Angriff"* [20].
Allerdings war nicht nur GitHub von der Attacke betroffen. Laut dem Unternehmen "Qihoo 360 Technology[12]" kam es innerhalb von sieben Tagen zu Angriffen auf 7131 unterschiedliche IP's. Darunter befinden sich auch Webseiten von bekannten Unternehmen aus den verschiedensten Bereichen, wie z.B. Google, Rockstargames und Kaspersky [35]. Für diese globale DDoS-Welle wurde eine Sicherheitslücke in der Software "Memcached" genutzt, welche in diesem Kapitel näher untersucht wird.

[12]Chinesisches Unternehmen, welches auf Netzwerksicherheit spezialisiert ist und einen DDoS Monitoring Service betreibt.

2.2.2 Verwendete Software

Für die eigenen Tests zur Bearbeitung dieser Sicherheitlücke wurden verschiedene Programme genutzt. Diese werden im Folgenden kurz beschrieben.

Name	**Beschreibung**
Kali Linux	Linux-Distribution (Betriebssystem)
Memcached	Memcached Version 1.5.6
Telnet	Telnet nutzt das gleichnamige Netzwerkprotokoll, um einen Datenaustausch über TCP zu ermöglichen. Es wird dazu genutzt, um mit dem lokalen Memcached Server zu kommunizieren.
Netcat	Netcat, abgekürzt "nc", ist ein Kommandozeilenwerkzeug für die Kommunikation über TCP- und UDP-Netzwerkverbindungen. Es wird dazu genutzt, um Kommandos über UDP an den Memcached Server zu übertragen.
Wireshark	Wireshark ist ein Tool, welches zur Aufzeichnung und Analyse des Netzwerkverkehrs dient. Es wird genutzt, um die übertragenen und empfangenen Netzwerkpakete, von und zu Memcached, zu analysieren.
Python	Programmiersprache, mit welcher kleinere Programme für die Tests erstellt werden. Version 2.7.15.
pymemcache	Python API, welche einen Memcached Client für Python bereitstellt. Diese wird in einem Testprogramm genutzt.

Tabelle 2.1: Übersicht der verwendeten Softwarekomponenten

2.2.3 Allgemeine Grundlagen

Memcached

Bei "Memcached" handelt es sich um eine Open-Source Software, welche im Jahr 2003 ursprünglich von Brad Fitzpatrick für die Webseite *livejournal.com* entwickelt wurde. Bis heute haben zu dem Projekt ca. 170 Mitwirkende etwas beigesteuert [5].
Memcached wird laut den Entwicklern folgendermaßen beschrieben: *"Free and open source, high-performance, distributed memory object caching system"* [5]. Es ist also im Prinzip eine simple Anwendung, die es ermöglicht, Daten im Arbeitsspeicher zu hinterlegen und wieder abzufragen. Die Daten werden hierbei jeweils über ein Schlüssel-Wert Paar beschrieben [5]. Hierbei kann es sich um beliebige Arten von Daten handeln. Memcached

selbst versteht keine Datenstrukturen, weshalb die Daten, über die jeweilige Programmiersprache, vor dem Speichern "vor-serialisiert" werden müssen [6]. Eine Memcached-API existiert für die folgenden Programmiersprachen: Python, Java, PHP, C/C#/C++, Perl Go, Ruby, JavaScript.
Der Vorteil von Memcached ist, dass die Zugriffszeiten auf die Daten sehr schnell sind, da sie sich im Arbeitsspeicher befinden. Gewöhnliche Datenbanksysteme (ausgenommen "In-Memory Datenbanken") speichern die Daten auf der Festplatte ab. Besitzt man also ein Anwendungsszenario, bei dem die Persistenz der Daten eine untergeordnete Rolle spielt und die Zugriffszeiten der entscheidende Faktor ist, so ist der Einsatz von Memcached als durchaus sinnvoll zu betrachten. Ein beliebter Einsatzzweck bei Webapplikationen ist hierbei zum Beispiel die Verwendung als Sitzungsspeicher [1].

Shodan

Bei Shodan handelt es sich um eine Suchmaschine, welche über die URL *https://www.shodan.io/* (Stand:28.05.2019) erreichbar ist. Sie sucht dabei, im Gegensatz zu Google, nicht nur nach Webseiten, sondern auch nach allen anderen Dingen (Geräten), welche aus dem Internet erreichbar sind. Shodan selbst bezeichnet sich auf ihrer Webseite als *"Search Engine for Security, Internet of Things"* und einige mehr. Mit Hilfe dieses Services lässt sich das Internet beispielsweise nach Servern durchforsten, welche einen bestimmten Port geöffnet haben. Dabei kann entweder eine eigene IP Adresse angeben werden, oder man lässt Shodan nach allen bisher gesammelten Datenbankeinträgen von Servern suchen. Diese Suchmaschine ist in sofern wichtig, da sie einen erheblichen Teil zum Erfolg der DDoS-Attacken beigetragen hat.

TCP vs. UDP

Zur Ausnutzung der Sicherheitslücke wurde das UDP Protokoll verwendet. Zum besseren Verständnis der folgenden Abschnitte sollen in diesem Kapitel die grundlegenden Unterschiede der zwei Netzwerkprotokolle, TCP und UDP, erläutert werden.
Beide Protokolle befinden sich auf der vierten Schicht (Transportschicht) des OSI-Modells.

TCP Bei TCP handelt es sich um ein verbindungsorientiertes Protokoll. Das bedeutet, dass über einen so genannten "Three-Way-Handshake" eine logische Verbindung zwischen Sender und Empfänger aufgebaut wird. Außerdem stellt TCP sicher, dass die Daten fehlerfrei und vollständig beim Empfänger eingegangen sind. Sollte dies nicht der Fall sein, so stellt TCP außerdem einen Mechanismus zur Verfügung, der dafür sorgt,

dass fehlende Daten vom Sender erneut geschickt werden. Aufgrund dieser Eigenschaften wird TCP auch als “zuverlässiges“ Protokoll bezeichnet.
Diese Merkmale haben allerdings zufolge, dass die Datenübertragung per TCP etwas zeitintensiver ist, als es beispielsweise bei UDP der Fall ist. Durch den Verbindungsaufbau, die Empfangsbestätigungen und andere Mechanismen, besitzt TCP einen relativ großen Overhead.

UDP UDP hingegen ist als eine einfach gehaltene und schnellere Alternative zu TCP anzusehen. Es wird auch als verbindungsloses und “unzuverlässiges“ Protokoll bezeichnet, da es keine Verbindung aufbaut, sondern die Daten direkt gesendet werden können. Außerdem existiert in diesem Protokoll kein Mechanismus, der dem Sender mitteilt, ob die Daten wirklich beim Empfänger angekommen sind (Empfangsbestätigung). Aufgrund dieser Eigenschaften ist UDP ein schlankes Protokoll, welches im Gegensatz zu TCP, einen relativ geringen Overhead an Daten besitzt. Ein beliebter Einsatzzweck für UDP sind beispielsweise Videostreams. Hierbei ist es für den Empfänger wichtiger, dass die Bilddaten schnell übertragen werden, als die Sicherstellung, dass alle Daten korrekt übertragen wurden. Sollten einzelne Bilder gelegentlich Fehler aufweisen, so ist das in diesem Szenario besser zu verkraften als eine insgesamt langsame Datenübertragung.

IP Spoofing

Unter “IP Spoofing“ bezeichnet man eine Technik, mit der man IP Pakete mit gefälschter Ursprungsadresse versendet.
Das Internet Protokoll (IP) befindet sich auf der dritten Schicht des OSI-Modells. Der Header eines jeden IP-Pakets ist in Abbildung 2.22 dargestellt. Dieser beinhaltet unter anderem die IP-Adresse des Absenders des Pakets (Quelle). Der Empfänger (Ziel) des Pakets weiß daraufhin von wem die Daten gekommen sind und an welche Adresse er seine Antwort schicken muss. Das Problem hierbei ist, dass der Absender eine beliebige Quell-IP-Adresse in den Header eintragen kann. Dies hat zur Folge, dass der Empfänger seine Antwort nicht an denjenigen schickt, von dem das Paket ursprünglich stammt, sondern an eine dritte Stelle.
Bei TCP basierten Verbindungen lässt sich diese Methode nicht so einfach einsetzen, da vor der eigentlichen Datenübertragung immer ein Verbindungsaufbau, über den “Three-Way-Handshake“, stattfinden muss. Allerdings sei an dieser Stelle erwähnt, dass es durchaus Angriffe auf TCP/IP gibt, welche sich IP Spoofing zu Nutze machen. Oftmals handelt es sich hierbei um so genannte “Man-in-the-middle“ Attacken, welche allerdings

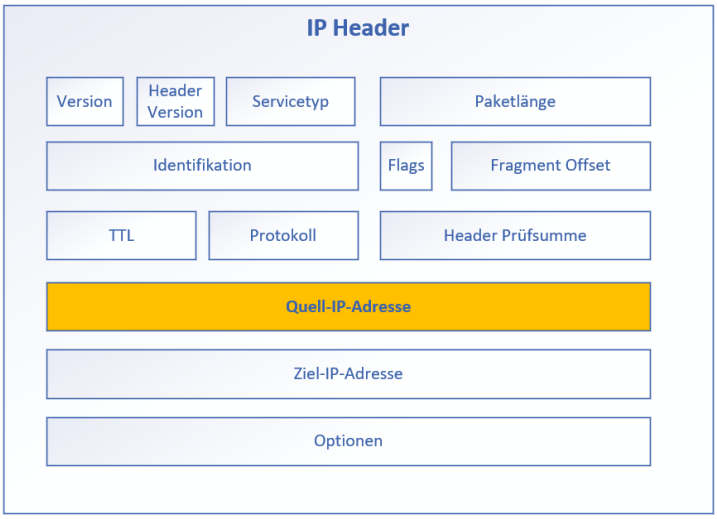


Abbildung 2.22: Aufbau des IP Headers

erst nach erfolgreichem Verbindungsaufbau zwischen Sender und Empfänger zum Tragen kommen. Auf Details hierzu wird in dieser Arbeit jedoch verzichtet.
Da bei UDP kein wirklicher Verbindungsaufbau stattfindet, lässt sich IP Spoofing in diesem Umfeld sehr einfach umsetzen. Der Sender eines UDP Pakets kann beim Versenden der Nachricht eine beliebige IP Adresse in den darüberliegenden IP Header schreiben. An diese Adresse schickt der Empfänger anschließend seine Antwort (siehe Abbildung 2.23).

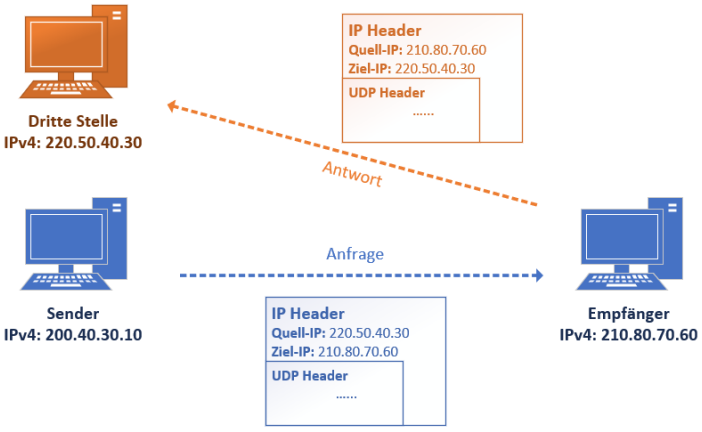


Abbildung 2.23: IP Spoofing über UDP

Einordnung der DDoS-Attacke

DDoS (Distributed Denial of Service) bezeichnet eine Technik, mit welcher mehrere (Distributed) Computer einen oder mehrere Computer, über bestimmte Verfahren, überlasten (Denial of Service). Ziel dieses Angriffs ist es, die entsprechenden IT-Geräte (oft Server) des Opfers so auszulasten, so dass diese für eine gewisse Zeit nicht mehr genutzt werden können. Dies kann unter anderem auch zu wirtschaftlichem Schaden bei den Opfern führen, wenn dabei geschäftskritische Komponenten, wie beispielsweise die Webseite eines Online Shops, getroffen werden.

DDoS-Attacken lassen sich auf sehr viele unterschiedliche Art und Weisen durchführen. Die Art von DDoS, welche sich durch über Memcached durchführen lässt, wird als "UDP-basierte Verstärkungsattacke" bezeichnet. "UDP-basiert" bedeutet dabei, dass für die Datenübertragung das Transportprotokoll UDP genutzt wird, welches im vorherigen Abschnitt beschrieben wurde. Der Begriff "Verstärkung" (eng. Amplification) wird verwendet, da die Menge der Daten, welche der Angreifer versendet (Anfragedatenmenge), kleiner sind als jene, die beim Opfer schlussendlich eintreffen (Antwortdatenmenge). In diesem Kontext wird auch vom so genannten *Verstärkungsfaktor* gesprochen. Dieser soll folgendermaßen definiert werden:

Definition 2.1: Verstärkungsfaktor

Bei DDoS-Attacken bezeichnet der Begriff *Verstärkungsfaktor* (V) den Faktor, um welchen sich die Antwortdatenmenge (a) gegenüber der Anfragedatenmenge (r) (beide in Bytes) erhöht:

$$V = a/r$$

für $a, r \in \mathbb{N}, a \geq r$

Da der Angreifer bei einem solchen Szenario nicht direkt Daten an das Opfer sendet, sondern es einem Dritten überlässt (in diesem Fall Memcached Servern), spricht man hier auch von einer "Reflexions"-Attacke (eng. Reflection). Dieser Begriff wird sehr häufig von einer Verstärkungs-Attacke impliziert, da ein Verstärkungsfaktor meistens nur über eine dritte Stelle ermöglicht erreicht werden kann.

2.2.4 Analyse

Am Anfang dieses Kapitels erfolgt zum besseren Verständnis eine grundlegende Beschreibung der DDoS-Attacke. Anschließend werden bestimmte Themen auf detaillierterer Ebene untersucht.

Beschreibung der DDoS Attacke

Hinweis: Zur besseren Lesbarkeit wird in diesem Abschnitt, in Bezug auf die vergangen DDoS-Attacken, die Vergangenheitsform gewählt. Bei entsprechender Konfiguration lässt sich diese Attacke jedoch heute noch genauso umsetzen.

Wie zu Beginn dieses Kapitels beschrieben, handelt es sich hier um eine Sicherheitslücke in der Software Memcached, welche dazu genutzt werden konnte, um DDoS-Attacken durchzuführen. Memcached war bis zum entsprechenden Sicherheitspatch standardmäßig nicht nur über TCP, sondern auch über UDP erreichbar. Außerdem existierten und existieren immer noch sehr viele Server, auf denen Memcached installiert ist und welche ungeschützt über das Internet erreichbar sind. Die Angreifer konnten sich die entsprechenden Server über die Suchmaschine Shodan zusammenstellen. War eine ausreichende Anzahl an Servern gefunden, so konnten die Angreifer bestimmte Kommandos (dazu mehr im nachfolgenden Abschnitt) an die jeweiligen Server schicken. Diese Kommandos wurden per UDP übermittelt und durch IP Spoofing (siehe Abschnitt “IP Spoofing“ in 2.2.3) wurden die Antworten an das entsprechende Opfer geschickt (siehe Abbildung 2.24). Dadurch, dass Memcached Kommandos bei ihrer Anfrage sehr klein sind, aber

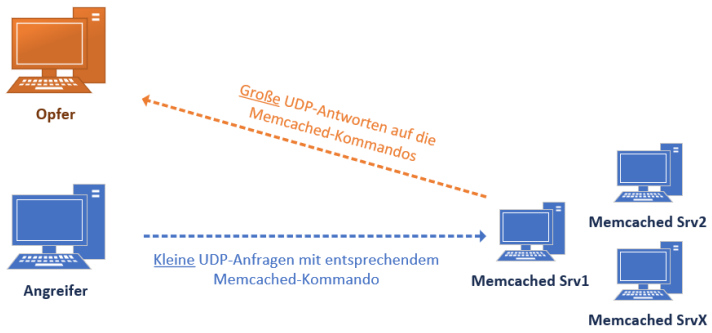


Abbildung 2.24: DDoS-Attacke über Memcached Server

die Antworten sehr groß werden können, konnte durch dieses Szenario, mit geringem Aufwand, eine sehr große Datenmenge erzeugt werden (siehe Angriff auf GitHub in Abschnitt 2.2.1).

Memcached Kommandos

Wie bereits beschrieben, handelt es sich bei Memcached um eine Software, welche Daten im Arbeitsspeicher in Schlüssel-Wert Form abspeichert. Die wichtigsten Kommandos hierfür sollen in diesem Abschnitt kurz erläutert werden. Alle anderen Kommandos, welche außerdem in der jeweiligen Rubrik zur Verfügung stehen, werden nur benannt[13]. Für die folgenden Beispiele wird dafür eine Telnet-Verbindung auf den lokal installierten Memcached-Server (Port 11211) genutzt.

```
jboe@kali:~$ telnet localhost 11211
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

Abbildung 2.25: Telnet Verbindung

1. Speicher-Kommandos

Um Daten speichern zu können, existiert das so genannte "Set-Kommando". Dieses be-

```
set foo 42 3600 3
bar
STORED
```

Abbildung 2.26: Set-Kommando

nötigt die folgenden Parameter: 1. Schlüsselname 2. Individuelles Flag 3. Ablaufzeit 4. Länge des dazugehörigen Wertes in Bytes. Im zweiten Schritt folgt die Angabe des Wertes, den man zu diesem Schlüssel hinterlegen möchte.

Weitere Kommandos in dieser Rubrik: *add*, *replace*, *append*, *prepend*, *cas (Check and Set)*.

2. Abfrage-Kommandos

Möchte man die Daten wieder abrufen, so nutzt man das "Get-Kommando". Dieses benö-

```
get foo
VALUE foo 42 3
bar
END
```

Abbildung 2.27: Get-Kommando

tigt den Namen des jeweiligen Schlüssels und gibt den zugehörigen Wert, das hinterlegte

[13] Details hierzu können in der offiziellen Dokumentation der Kommandos entnommen werden: https://github.com/memcached/memcached/wiki/Commands#storagecommands (zuletzt abgerufen am: 10.05.2019)

Flag und die Größe an den Aufrufer zurück. Es können hierbei auch mehrere Schlüssel gleichzeitig abgefragt werden (Trennung der Schlüsselnamen durch Leerzeichen).

Weitere Kommandos in dieser Rubrik: *gets*, *delete*, *incr/decr*.

3. Statistik-Kommandos

Über das "Stats"-Kommando lässt sich eine Statistik zu dem installierten Memcached-Server abrufen. Durch dieses erhält man unter anderem Informationen über gespeicherte

```
stats
STAT pid 2192
STAT uptime 240
STAT time 1558996831
STAT version 1.5.6
STAT libevent 2.1.8-stable
STAT pointer_size 64
STAT rusage_user 0.053301
STAT rusage_system 0.017767
STAT max_connections 1024
STAT curr_connections 5
STAT total_connections 6
STAT rejected_connections 0
STAT connection_structures 6
STAT reserved_fds 20
STAT cmd_get 1
STAT cmd_set 1
STAT cmd_flush 0
STAT cmd_touch 0
STAT get_hits 1
STAT get_misses 0
STAT get_expired 0
STAT get_flushed 0
STAT delete_misses 0
STAT delete_hits 0
STAT incr_misses 0
STAT incr_hits 0
STAT decr_misses 0
STAT decr_hits 0
STAT cas_misses 0
STAT cas_hits 0
STAT cas_badval 0
STAT touch_hits 0
STAT touch_misses 0
STAT auth_cmds 0
STAT auth_errors 0
STAT bytes_read 40
STAT bytes_written 34
```

Abbildung 2.28: Stats-Kommando

Datenmengen und die Anzahl an bestimmten Kommandos, die bei dieser Installation bis zum jetzigen Zeitpunkt genutzt wurden. In Abbildung 2.28 ist beispielsweise zu erkennen , dass in diesem Beispiel jeweils einmal das Get- und Set-Kommando ausgeführt wurde.

Weitere Kommandos in dieser Rubrik: *stats items*, *stats slabs*, *stats sizes*.

Unabhängig von diesen drei Kategorien existiert noch ein Kommando, welches dazu führt, dass alle gespeicherten Schlüssel-Wert Paare ablaufen und nicht mehr zur Verfügung stehen: *flush_all*.

Memcached Verbindungsarten und offene Server

Bei einer Standardkonfiguration von Memcached (bis einschließlich Version 1.5.5) ist der Port 11211 über TCP als auch über UDP erreichbar [12]. Memcached-Server sind im Regelfall dazu gedacht, nur über das interne Netzwerk erreichbar zu sein, um dort (oft vertrauliche) Daten im Arbeitsspeicher zu halten. Allerdings hat 2018 eine Abfrage bei der Suchmaschine Shodan ergeben, dass ca. 88.000 Memcached-Server offen über das Internet erreichbar sind (Abbildung 2.29). Ob ein Server dabei auch über UDP erreichbar

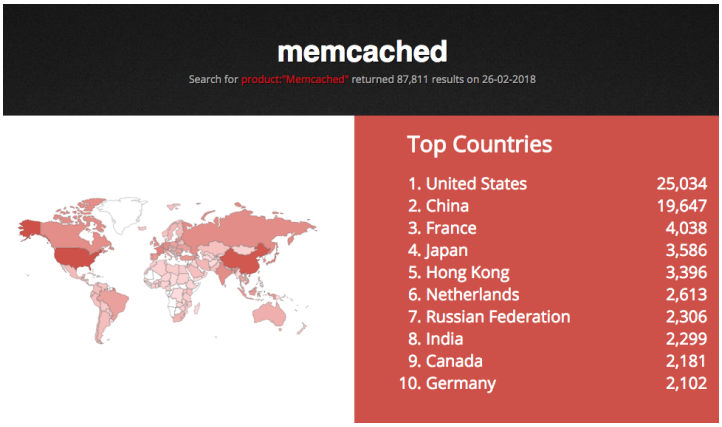


Abbildung 2.29: Verteilung der offenen Memcached-Server auf Länder [26]

ist, lässt sich sehr einfach über das "Stats-Kommando" ermitteln:

```
echo -en "\x00\x00\x00\x00\x00\x01\x00\x00stats\r\n" | netcat -
    u 192.168.45.67 11211
```

Erläuterung: Bei den 8 Bytes vor dem eigentlichen Kommando handelt es sich um einen von Memcached definierten "Frame Header", welcher jedem Kommando vorangestellt werden muss. Anschließend folgt das jeweilige Kommando, in diesem Fall "stats". Der Abschluss einer Anfrage erfolgt mit den 2 Bytes "\r\n". Für nähere Informationen hierzu sei auf die Protokollbeschreibung von Memcached verwiesen: [7].
Wenn diese Anfrage erfolgreich war, liefert der Server Informationen über die Memcached Installation zurück (vgl. Abbildung 2.28).
Die Suchmaschine Shodan und diese kurze Testabfrage sind ausreichend, um sich als Angreifer eine Liste mit Servern zusammenzustellen, welche für eine DDoS-Attacke genutzt werden können.

Untersuchung der Datenmengen und Verstärkungsfaktoren

Das Unternehmen SISSDEN [10] sowie Netlab360 [35] haben im Rahmen ihrer Recherchen zu den DDoS-Attacken 2018 so genannte "Honeypots" zur Verfügung gestellt. Das bedeutet, dass eigene Memcached Server offen im Internet betrieben wurden, mit dem Ziel, dass einige Angreifer diese für ihre Attacken nutzen. Dabei konnten unter anderem Informationen über die IP Adressen der Angreifer und auch die für die Angriffe genutzten Memcached-Kommandos ermittelt werden. Dabei hat sich herauskristallisiert, dass in den meisten Fällen das **stats** Kommando, aber auch gelegentlich das **get** Kommando genutzt wurde (vgl. [35]).
Diese beiden Kommandos sollen in den nächsten Abschnitten in Bezug auf die Datenmengen näher untersucht werden. Zur Veranschaulichung der Tests sind hierbei gelegentlich Bildschirmaufnahmen von Wireshark abgebildet. Diese beinhalten allerdings immer die Gesamtgröße der jeweiligen Pakete. Für die Berechnung der Verstärkungsfaktoren werden jedoch nur die reinen Nutzdaten herangezogen. Was in dieser Arbeit jeweils als Nutzdaten definiert wird, ist am Anfang des entsprechenden Kommandos beschrieben.

Stats

Definition Nutzdaten:
Anfrage: 5 Bytes "stats", da keine Parameter vorhanden (fix).
Antwort: Alle Daten, welche das Stats-Kommando über die Memcached Installation zurückliefert, ohne zusätzliche Informationen wie die Markierung des Endes der Empfangenen Daten etc. (variabel).

Die "Stats-Anfrage" über UDP beinhaltet Nutzdaten von 5 Bytes und ist bei der Übertragung insgesamt 57 Bytes groß[14]. Die Antwort hingegen liefert eine sehr viel Größere Menge an Daten zurück. In Abbildung 2.30 ist dabei zu erkennen, dass das Stats-Kommando eine Antwortgröße von 2007 (Nutzdaten: 1959) Bytes besitzt. Daraus ergibt

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	35905	11211	MEMCACHE	57	MEMCACHE Continuation
2	0.000195181	11211	35905	MEMCACHE	1442	MEMCACHE Continuation
3	0.000204265	11211	35905	MEMCACHE	565	MEMCACHE Continuation

Abbildung 2.30: Wireshark: Stats-Kommando über UDP

sich bei diesem Kommando ein Verstärkungsfaktor von **391,8** (1959 Bytes / 5 Bytes).

[14]Ethernet: 14 Bytes, IP-Header: 20 Bytes, UDP-Header: 8, Memcached UDP-Header: 8 Bytes, Abschluss: 2 Bytes

Hierzu sei angemerkt, dass sich dieser Verstärkungsfaktor schon ohne vorherige Präparation des jeweiligen Memcached-Servers erreichen lässt (d.h. es müssen keine Daten im Vornherein hinterlegt werden).

Get

Definition Nutzdaten:

Anfrage: Nur Parameter + Trennzeichen (variabel).

Antwort: Alle Daten, welche das Get-Kommando zu den jeweiligen Schlüsseln zurückliefert, ohne zusätzliche Informationen wie die Markierung des Endes der Empfangenen Daten etc. (variabel).

Das Get-Kommando nimmt einen oder mehrere Schlüssel entgegen, zu denen die entsprechenden Daten abgerufen werden sollen. Zunächst soll die Antwortdatenmenge eines Get-Kommandos mit **einem** Schlüssel untersucht werden. Dazu wird ein Datenpaar auf dem Memcached-Server angelegt:

```
set a 0 3600 3
foo
```

Dieses wird anschließend über UDP wieder abgerufen:

```
echo -en "\x00\x00\x00\x00\x00\x01\x00\x00get␣a\r\n" | netcat -
    u 192.168.45.67 11211
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	60234	11211	MEMCACHE	57	MEMCACHE Continuation
2	0.000127528	11211	60234	MEMCACHE	73	MEMCACHE Continuation

Abbildung 2.31: Wireshark: Größe der UDP Antwort (Versuch 1)

In Abbildung 2.31 ist dabei zu erkennen, dass die Antwort insgesamt 73 Bytes groß ist. 3 Bytes davon sind der eigentliche Wert “foo“.

Es ist zu vermuten, dass sich mit der Erhöhung der Datenmenge zu einem Schlüssel ein sehr hoher Verstärkungsfaktor erreichen lässt. Hierzu sollen in diesem Abschnitt einige Überlegungen und Tests stattfinden.

Die maximale Größe der Daten eines Schlüssels beträgt, in der Standardkonfiguration von Memcached, 1 MByte[15]. Das würde bedeuten, dass sich durch eine 2 Byte große Get-Anfrage (kleinste Form des Kommandos), ca. 1 MByte an Antwortdaten erzeugen

[15] siehe: https://docs.oracle.com/cd/E17952_01/mysql-5.6-en/ha-memcached-faq.html#faq-memcached-max-object-size , zuletzt abgerufen am: 13.05.2019

lassen könnte. Um dies nachvollziehen zu können, wurde ein Python Skript entwickelt, mit welchem über die Memcached API (per TCP) Daten auf dem Server gespeichert werden können. Hierzu sei erwähnt, dass es über diese API nicht möglich war, genau 1 MByte, also 1.000.000 Bytes, an Daten zu einem Schlüssel zu hinterlegen. Im Rahmen dieser Arbeit konnten die Gründe dafür nicht ermittelt werden und es wurde aufgrund dessen ein etwas geringerer Wert genommen (999.902), mit welchem es über diese API funktioniert.

```
from pymemcache.client.base import Client
client = Client(('localhost', 11211))
KEY = "i"
VALUES = "j" * 999.902
client.set(KEY, VALUES)
```

Einschub 2.1: Set-Kommando über UDP

An dieser Stelle sei angemerkt, dass das Hinterlegen von Daten auf dem Server in dieser Größenordnung über UDP nicht möglich ist. Laut der Protokollbeschreibung von Memcached darf ein Kommando (Anfrage) nur ein UDP-Datagramm umfassen, wobei die Antwort von Memcached jedoch mehrere Datagramme umfassen kann [7]. Das bedeutet, dass man über UDP höchstens die folgende Größe an Daten zu einem Schlüssel hinterlegen kann:
(In Bytes) Länge eines UDP-Datagramms (Header + Nutzdaten): $2^{16} - 1 = 65.535$
65.535 - 20 (IPv4 Header) - 8 (UDP Header) - 8 (Memcached UDP Header) - 3 (Set) - 2 (Abschluss) = **65494**.

Ruft man diese Daten über UDP wieder ab, so ist in Abbildung 2.32 zu erkennen, dass die Antwort 718 Pakete der Größe 1442 Bytes und ein Paket der Größe 521 Bytes umfasst. Das erste Paket ist die Get-Anfrage und zählt nicht zu der Antwort.

No.	Time	Source	Destination	Protocol	Length	Info
714	0.007732090	11211	44368	MEMCACHE	1442	MEMCACHE Continuation
715	0.007739159	11211	44368	MEMCACHE	1442	MEMCACHE Continuation
716	0.007746219	11211	44368	MEMCACHE	1442	MEMCACHE Continuation
717	0.007753204	11211	44368	MEMCACHE	1442	MEMCACHE Continuation
718	0.007759886	11211	44368	MEMCACHE	1442	MEMCACHE Continuation
719	0.007766879	11211	44368	MEMCACHE	1442	MEMCACHE Continuation
720	0.007774432	11211	44368	MEMCACHE	521	MEMCACHE Continuation

Abbildung 2.32: Wireshark: Größe der UDP Antwort (Versuch 2)

Rechnung 2.1

Berechnung der Nutzdaten:
Hinterlegte Datenmenge (siehe Python Skript): 999.902 Bytes
Antwort: Pro Paket jeweils 42 Bytes für Ethernet, UDP- und IP-Header. 8 Bytes für den Memcached Header
→ 718 * 1392 + 1 * 471 = 999.927
→ 999.927 - 19 (zusätzliche Informationen im ersten Antwortpaket) - 6 (Markierung des Endes im letzten Paket) = **999.902** ✓(Nutzdaten)

Rechnung 2.1 bestätigt, dass genau die Anzahl an Nutzdaten zurückgeliefert wurde, die zuvor über das Python Programm in das System eingepflegt wurde. In diesem Fall wurde also bereits ein Verstärkungsfaktor von **499.501** (999.902 Bytes / 2 Bytes) erzielt. Der Verstärkungsfaktor der reinen Nutzdaten beim Get-Kommando lässt sich allgemein wie folgt definieren (bei Schlüsselnamen mit nur **einem** Zeichen):

Definition 2.2: Verstärkungsfaktor Get-Kommando

Wenn k die Anzahl der abzurufenden Schlüssel ist und g die Größe der Daten eines k in Bytes beschreibt, ergibt sich auf Grundlage von Definition 2.1 für $a = g * k$ und $r = 2 * k$ (pro Schlüssel 2 Bytes wegen Leerzeichen), weshalb für den Verstärkungsfaktor in diesem Fall Folgendes gilt:

$$V_{Get} = \frac{g * k}{2 * k} = \frac{1}{2}g$$

für $g \in \mathbb{N}, 2 \leq g \leq 1000000$

Wie Definition 2.2 zu entnehmen ist, ist der Verstärkungsfaktor bei diesem Kommando unabhängig von der Anzahl der verwendeten Schlüssel und hängt nur von der Größe der Daten eines Schlüssels (g) ab. Da die maximale Größe, wie zuvor erläutert, bei 1 MByte liegt, ist zu erkennen, dass man sich dem maximalen Verstärkungsfaktor durch den vorherigen Versuch (siehe Rechnung 2.1) bereits sehr gut angenähert und auch praktisch nachgewiesen hat. In der Theorie sollte es also möglich sein, wenn man es schafft die vollen 1 MByte pro Schlüssel zu hinterlegen, das folgende Maximum zu erreichen:
$V_{Get(max)} = \frac{g}{2} = \frac{1000000}{2} = 500.000$

An dieser Stelle sei jedoch angemerkt, dass der Verstärkungsfaktor keinen Hinweis darauf liefert, wie hoch die größtmögliche Datenmenge ist, die man einen Memcached Server erzeugen lassen kann.
Da in der Memcached Dokumentation keine Informationen über die Beschränkung der Parameter des Get-Kommandos zu finden sind, sollte es theoretisch möglich sein, ein komplettes UDP-Datagramm (65494 Bytes, siehe Einschub 2.1) mit Schlüsselnamen zu füllen. Da sich zwischen den Schlüsselnamen immer ein Leerzeichen befinden muss, ergibt sich für die maximale Anzahl an Schlüsseln: 65494/2 = 32.747.
Daraus folgt für die Anfragedatenmenge $r = 2 * k = 2\ \text{Bytes} * 32747 = 65494$ Bytes und für Antwortdatenmenge $a = g * k = 1000000\ \text{Bytes} * 32747 = 32.747.000.000$ Bytes.
Dies würde bedeuten, dass sich durch eine ≈65 KByte große Anfrage, eine ≈32 GByte große Antwortdatenmenge erzeugen lassen könnte.
Um ein Get-Kommando über UDP mit beliebig vielen Parametern versehen zu können wurde folgendes Python Skript entwickelt:

```
import socket

UDP_IP = "127.0.0.1"
UDP_PORT = 11211
key = " k"*1000 //Anzahl der Schlüssel
MESSAGE = "\x00\x00\x00\x00\x00\x01\x00\x00get" + key + "\r\n"

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(bytes(MESSAGE.encode("utf-8")), (UDP_IP, UDP_PORT))
```

Ein Versuch einer Abfrage mit k=1000 auf dem lokalen Memcached-Server hat allerdings bereits dazu geführt, dass die genutzte virtuelle Maschine, aufgrund der hohen Auslastung, nicht mehr zu bedienen war. Jedoch konnte in Wireshark beobachtet werden, dass die Anzahl der empfangenen Pakete von Memcached immer weiter zugenommen hat. Es lässt sich daher also vermuten, dass von Memcached keine Restriktion bei den Parametern des Get-Kommandos existiert und sich eine solch hohe Datenmenge durchaus erreichen lassen kann. Zur besseren Veranschaulichung der Möglichkeiten, welche sich hierdurch ergeben, kann man sich beispielsweise Szenario 2.1 vor Augen führen.

Szenario 2.1: Beanspruchungszeit

Ein Memcached Server befindet sich in einem Rechenzentrum mit einer relativ guten Internetanbindung (Upload: 100 MBit/s), welche voll genutzt werden kann. Dieser Server bräuchte für die Übertragung von 32 GByte im Idealfall ca. 43 Minuten (32000 MByte / 12,5 MByte/s). Das bedeutet, dass man einen solchen Server, mit einer einzelnen 65 KByte großen Anfrage, für eine knappe Dreiviertelstunde dazu bringen kann, pausenlos Daten an das Opfer zu übertragen.

Sieht man sich mit diesem Kenntnisstand die Attacke auf GitHub mit 1,35 TBit/s (siehe Abschnitt 2.2.1) noch einmal an, so kann man ungefähr abschätzen, wie viele Memcached Server für diesen Angriff gleichzeitig in Benutzung waren:

Szenario 2.2: GitHub Angriff

Man nehme auch in diesem Fall wieder idealisiert an, dass jeder verfügbare Memcached Server einen Upload von 100 MBit/s voll zur Verfügung hat. Durch die kleinen Anfragen und den sich daraus ergebenden, sehr hohen, Verstärkungsfaktoren ist es ein leichtes Unterfangen, eine hohe Anzahl von derartigen Servern “zeitgleich“ für eine gewisse Zeit auszulasten (siehe Szenario 2.1). Angenommen, jeder dieser Server kann zur gleichen Zeit die vollen 12,5 MByte/s an GitHub übertragen, so wären ca. 13.500 Server dafür notwendig. Erläuterung:
1,35 TBit/s = 168.750 MByte/s
168.750 MByte/s / 12,5 MByte/s = 13.500
Wenn man nun davon ausgeht, dass diese Server schon zuvor mit den entsprechenden Daten versehen wurden (jeweils 1 MByte pro Server), müsste man als Angreifer nur noch folgendes bewältigen: An jeden dieser Server sollte in einem kleinen Zeitfenster eine 65 KByte große Anfrage geschickt werden, was bei 13.500 Servern einer Gesamtdatenmenge von ca. 900 MByte entspricht. Unter heutigen Voraussetzungen sollte dies kein Problem darstellen.

Anhand von den beiden beschrieben Szenarien lässt sich nun etwas besser nachvollziehen, wie eine solche große Attacke in etwa durchgeführt werden konnte. Es wurde zwar in diesen Fällen mit idealisierten Datenübertragungsraten gearbeitet und auch einige andere Rahmenbedingungen außer Acht gelassen. Jedoch sollte man noch einmal bedenken, dass zu diesem Zeitpunkt, neben den berechneten 13.500, noch ca. 74.500 weitere

Memcached Server offen zur Verfügung standen.
Es kann nicht mit Sicherheit gesagt werden, welche Memcached Kommandos in welcher Anzahl für die Attacke auf GitHub genutzt wurden. Allerdings lässt sich, durch die Erkenntnisse in dieser Arbeit, mit einer sehr hohen Wahrscheinlichkeit vermuten, dass vermehrt das Get-Kommando zum Einsatz gekommen ist.

Vergleich zu anderen DDoS Attacken Das "Department of Homeland Security" in den USA hat auf ihrer offiziellen Webseite unter der Rubrik "UDP-Based Amplification Attacks" die jeweiligen Verstärkungsfaktoren der verschiedenen Protokolle aufgelistet (siehe Abbildung 2.33). Darunter ist als bekannter UDP-Vertreter beispielsweise auch

Protocol	Bandwidth Amplification Factor	Vulnerable Command
DNS	28 to 54	see: TA13-088A [4]
NTP	556.9	see: TA14-013A [5]
SNMPv2	6.3	GetBulk request
NetBIOS	3.8	Name resolution
SSDP	30.8	SEARCH request
CharGEN	358.8	Character generation request
QOTD	140.3	Quote request
BitTorrent	3.8	File search
Kad	16.3	Peer list exchange
Quake Network Protocol	63.9	Server info exchange
Steam Protocol	5.5	Server info exchange
Multicast DNS (mDNS)	2 to 10	Unicast query
RIPv1	131.24	Malformed request
Portmap (RPCbind)	7 to 28	Malformed request
LDAP	46 to 55	Malformed request [6]
CLDAP [7]	56 to 70	—
TFTP [23]	60	—
Memcached [25]	10,000 to 51,000	—

Abbildung 2.33: UDP-Verstärkungsfaktoren im Überblick [28]

DNS, mit einem Verstärkungsfaktor von 28-54, zu finden. Bei Memcached wird der Verstärkungsfaktor auf bis zu 51.000 geschätzt, was durch die vorherigen Versuche in dieser Arbeit, als absolut realistisch erscheint. Woher dieser Wert stammt und über welche Kommandos dieser ermittelt wurde, ist leider unklar. Tatsache ist, dass in dieser Arbeit ein weitaus höherer Verstärkungsfaktor erreicht bzw. errechnet werden konnte (500.000). Jedoch sollte man hierbei auch bedenken, dass dieser Wert auf Grundlagen von selbst

definierten Nutzdaten basiert. Abgesehen davon, ist ein solch hoher Wert (51.000 oder auch 500.000) im Vergleich zu den allen bisher bekannt geworden DDoS-Attacken dieser Art, jedoch eine absolute Ausnahme. Dies verdeutlicht auch noch einmal, welches Gefahrenpotential sich durch diese Sicherheitslücke ergeben hat.

Patches

Hinweis: Die folgenden Informationen wurden aus den Realease Notes von Memcached (ReleaseNotes156 in [6]) entnommen.

Die Schließung der Sicherheitslücke erfolgte am 27. Februar 2018, also einen Tag vor dem Angriff auf GitHub. Die zuständigen Entwickler wurden schon bereits zu diesem Zeitpunkt über Angriffe dieser Art auf andere Ziele informiert.

Zur Behebung der Lücke wurde ein Rundumschlag auf das UDP Protokoll ausgeübt, indem die UDP Funktionalität in Memcached standardmäßig deaktiviert wurde. Als Grund dafür wird unter anderem angegeben, dass UDP in den letzten Jahren bei Memcached Nutzern immer mehr an Beliebtheit verloren hat und in den meisten Fällen nur noch TCP genutzt wird.

2.2.5 Fazit

Im Regelfall verbindet man den Begriff “Sicherheitslücke“ mit irgendeiner Form von fehlerhaften Implementierungen bzw. mit dem Begriff “Bug“. Dieses Beispiel macht allerdings deutlich, dass es sich bei einer Sicherheitslücke nicht immer um derartige Dinge handeln muss. Auch beabsichtige und fehlerfrei funktionierende Module einer Software, wie in diesem Beispiel die Kommunikation über UDP, können zu drastischen Folgen führen, welche in der Entwicklung so nicht bedacht worden sind.

Welche Möglichkeiten sich durch diese Sicherheitslücke ergeben haben, wurde einerseits durch das einführende Praxisbeispiel (DDoS Attacke auf GitHub) und andererseits im Abschnitt “Untersuchung der Datenmengen und Verstärkungsfaktoren“ versucht deutlich zu machen. Es konnte dadurch zu der bisher größten aufgezeichneten DDoS-Attacke in der Geschichte des Internets kommen. Die Verstärkungsfaktoren, welche durch Memcached erreicht werden konnten, haben alle bisher dagewesenen und vergleichbare Angriffe über UDP um ein vielfaches übertroffen (siehe Abbildung 2.33). Als Ursache für diese Sicherheitslücke würde man wahrscheinlich im ersten Gedanken die Entwickler als Verantwortliche sehen. Allerdings darf nicht vergessen werden, dass Memcached nie dafür vorgesehen war, öffentlich und ohne Schutzmaßnahmen (Firewalls etc.) über das Internet erreichbar zu sein. Umso erstaunlicher ist es also, dass im Jahr 2018 ca. 88.000

Memcached Server ungeschützt im Internet zur Verfügung standen. Als Gründe dafür lässt sich folgendes vermuten: Entweder wurden die Betreiber, beispielsweise über die entsprechende Dokumentation von Memcached, nicht ausreichend darüber informiert oder es wurde schlicht übersehen bzw. im Extremfall ignoriert.
Unabhängig von möglichen Anschuldigungen hat sich bei der Bearbeitung dieser Sicherheitslücke jedoch der folgende Knackpunkt herauskristallisiert: Es wurde eine Software für einen bestimmten Einsatzzweck entwickelt (Einsatz im internen Netzwerk), welche jedoch, bei entsprechender Netzwerkkonfiguration, auch für einen anderen Einsatzzweck genutzt werden konnte (Einsatz über das Internet). In einem solchen Szenario ist es oft so, dass der andere, nebensächliche Einsatzzweck bei der Entwicklung und bei den Tests, etwas aus den Augen verloren wird. Allerdings hätte sich diese Sicherheitslücke auch durch herkömmliche Softwaretests nicht entdeckt lassen können. Eigener Einschätzung nach war es vielmehr so, dass der andere Einsatzzweck bei der Entwicklung zu wenig Beachtung gefunden hat und man sich aufgrund dessen zu wenig Gedanken über kritische Aspekte machen konnte.
Zusammenfassend ist zu sagen, dass die Kombination aus UDP und vielen verfügbaren Servern im Internet, sehr häufig ein beträchtliches Gefahrenpotential für derartige DDoS-Attacken beinhaltet.

Literaturverzeichnis

[1] *Amazon Web Services, Inc.* URL https://aws.amazon.com/de/memcached/. Zuletzt abgerufen am 03.04.2019.

[2] Smb - server message block. *Elektronik-Kompendium*. URL https://www.elektronik-kompendium.de/sites/net/2101131.htm. Zuletzt abgerufen am 24.03.2019.

[3] xattr - extended attributes. *Linux Programmer's Manual*. URL http://man7.org/linux/man-pages/man7/xattr.7.html. Zuletzt abgerufen am 27.03.2019.

[4] [ms-cifs]: Common internet file system (cifs) protocol. *Microsoft Corporation*. URL https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-cifs/d416ff7c-c536-406e-a951-4f04b2fd1d2b. Zuletzt abgerufen am 24.03.2019.

[5] *Memcached*, . URL https://memcached.org/. Zuletzt abgerufen am 03.04.2019.

[6] *Memcached GitHub Wiki*, . URL https://github.com/memcached/memcached/wiki/Overview. Zuletzt abgerufen am 03.04.2019.

[7] *Memcached GitHub Protokollbeschreibung*, . URL https://github.com/memcached/memcached/blob/master/doc/protocol.txt. Zuletzt abgerufen am 03.04.2019.

[8] History of smb. *Ryussi Technologies*, 2016. URL https://www.mosmb.com/history-of-smb/. Zuletzt abgerufen am 24.03.2019.

[9] *Wikipedia*, 2017. URL https://upload.wikimedia.org/wikipedia/en/1/18/Wana_Decrypt0r_screenshot.png. Zuletzt abgerufen am 24.03.2019.

[10] Understanding memcache ddos attacks. *SISSDEN Blog*, 2018. URL https://sissden.eu/blog/understanding-memcache-ddos-attacks. Zuletzt abgerufen am 24.03.2019.

[11] V. Brandon. Gallery: 10 major organizations affected by the wannacry ransomware attack. *Techrepublic*, 2017. URL https://www.techrepublic.com/pictures/gallery-10-major-organizations-affected-by-the-wannacry-ransomware-attack/4/. Zuletzt abgerufen am 24.03.2019.

[12] BSI. Offene memcached-server. *Bundesamt fuer Sicherheit in der Informationstechnik*, . URL https://www.bsi.bund.de/DE/Themen/Cyber-Sicherheit/Aktivitaeten/CERT-Bund/CERT-Reports/HOWTOs/Offene-Memcached-Server/Offene-Memcached-Server_node.html. Zuletzt abgerufen am 24.03.2019.

[13] BSI. Ransomware. *Bundesamt fuer Sicherheit in der Informationstechnik*, . URL https://www.bsi-fuer-buerger.de/BSIFB/DE/Risiken/Schadprogramme/Ransomware/ransomware_node.html. Zuletzt abgerufen am 24.03.2019.

[14] S. Dillon. Doublepulsar initial smb backdoor ring 0 shellcode analysis. . URL https://zerosum0x0.blogspot.com/2017/04/doublepulsar-initial-smb-backdoor-ring.html. Zuletzt abgerufen am 27.03.2019.

[15] S. Dillon. Eternal exploits: Reverse engineering of fuzzbunch and ms17-010. *DEFCON*, . URL https://media.defcon.org/DEF%20CON%2026/DEF%20CON%2026%20presentations/zerosum0x0/. Zuletzt abgerufen am 27.03.2019.

[16] S. Dillon. Eternal exploits: Reverse engineering of fuzzbunch and ms17-010. *DEFCON 2018*, . URL https://www.youtube.com/watch?v=HsievGJQG0w. Zuletzt abgerufen am 27.03.2019.

[17] S. Dillon. Ms17-010 in 2018? *Derbycon 2018*, . URL https://www.youtube.com/watch?v=ZHWidYrEgNM. Zuletzt abgerufen am 27.03.2019.

[18] S. Dillon and D. Davis. Eternalblue: Exploit analysis and port to microsoft windows 10. *RiskSense Threat Research*. URL https://jennamagius.keybase.pub/EternalBlue_RiskSense-Exploit-Analysis-and-Port-to-Microsoft-Windows-10.pdf. Zuletzt abgerufen am 27.03.2019.

[19] Foreign and C. Office. Foreign office minister condemns north korean actor for wannacry attacks. *UK Government*. URL https://www.gov.uk/government/news/foreign-office-minister-condemns-north-korean-actor-for-wannacry-attacks. Zuletzt abgerufen am 24.03.2019.

[20] H. Gierow. Github uebersteht bislang staerksten ddos-angriff. *Golem*. URL `https://www.golem.de/news/akamai-github-uebersteht-bislang-staerksten-ddos-angriff-1803-133105.html`. Zuletzt abgerufen am 03.04.2019.

[21] N. Grossman. Eternalblue - everything there is to know. *Checkpoint Research*. URL `https://research.checkpoint.com/eternalblue-everything-know/`. Zuletzt abgerufen am 27.03.2019.

[22] L. Hay Newman. The leacked nsa spy tool what hacked the world. *Wired*, 2018. URL `https://www.wired.com/story/eternalblue-leaked-nsa-spy-tool-hacked-world/`. Zuletzt abgerufen am 24.03.2019.

[23] J. Kennedy and M. Satran. Microsoft smb protocol and cifs protocol overview. *Microsoft Corporation*, 2018. URL `https://docs.microsoft.com/en-us/windows/desktop/fileio/microsoft-smb-protocol-and-cifs-protocol-overview`. Zuletzt abgerufen am 24.03.2019.

[24] S. Kottler. February 28th ddos incident report. *The GitHub Blog*. URL `https://github.blog/2018-03-01-ddos-incident-report/`. Zuletzt abgerufen am 03.04.2019.

[25] P. Kulkarni, S. Patil, P. Kadam, and A. Dolas. Eternalblue: A prominent threat actor of 2017-2018. *VIRUS BULLETIN*, 2018. URL `https://www.virusbulletin.com/uploads/pdf/magazine/2018/201806-EternalBlue.pdf`. Zuletzt abgerufen am 24.03.2019.

[26] M. Majkowski. Memcrashed - major amplification attacks from udp port 11211. *Cloudflare Blog*, 2018. URL `https://blog.cloudflare.com/memcrashed-major-amplification-attacks-from-port-11211/`. Zuletzt abgerufen am 24.03.2019.

[27] E. McCall. Eternalblue: Exploit analysis and beyond. *InfoQ*, 2018. URL `https://www.infoq.com/presentations/eternalblue`. Zuletzt abgerufen am 24.03.2019.

[28] U. D. of Homeland Security. Alert (ta14-017a) udp-based amplification attacks. *CISA - Cyber Infrastructure*, 2018. URL `https://www.us-cert.gov/ncas/alerts/TA14-017A`. Zuletzt abgerufen am 14.05.2019.

[29] M. Russinovich, D. A. Solomon, and A. Ionescu. *Windows Internals, Part 1*. Microsoft Press, Washington, 6 edition, 2012. ISBN 978-0-7356-4873-9.

[30] R. Sharpe. Just what is smb? URL `https://www.samba.org/cifs/docs/what-is-smb.html#Copying`. Zuletzt abgerufen am 24.03.2019.

[31] M. Spaenhoff. Grundlagen von samba. 2002. URL `http://www.it-academy.cc/article/28/Grundlagen+von+Samba.html`. Zuletzt abgerufen am 24.03.2019.

[32] Trendmicro. Massive wannacry/wcry ransomware attack hits various countries. *Trendmicro*, 2017. URL `https://blog.trendmicro.com/trendlabs-security-intelligence/massive-wannacrywcry-ransomware-attack-hits-various-countries/`. Zuletzt abgerufen am 24.03.2019.

[33] W. Wang. eternalblue_exploit7.py. *github*, . URL `https://github.com/worawit/MS17-010/blob/master/eternalblue_exploit7.py`. Zuletzt abgerufen am 27.03.2019.

[34] W. Wang. Bug.txt. *github*, . URL `https://github.com/worawit/MS17-010/blob/master/BUG.txt`. Zuletzt abgerufen am 27.03.2019.

[35] X. Yang. Memcache udp reflection amplification attack ii: The targets, the sources and breakdowns. *Qihoo 360 Technology Blog*. URL `https://blog.netlab.360.com/memcache-udp-reflection-amplification-attack-ii-the-targets-the-sources-and-breakdowns-en/`. Zuletzt abgerufen am 03.04.2019.

Erklärung

Hiermit versichere ich, dass ich

. . . . . .

Vorname, Name; Matrikelnummer

die vorliegende Arbeit mit dem Titel

. . . . . .

Titel der Arbeit

selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet, sowie wörtliche und sinngemäße Zitate gekennzeichnet habe.

Ronsberg, den 30. Mai 2019

. . . . . .

Unterschrift des Verfassers