Inhaltsverzeichnis

1 – Angegriffene Windows-Software 2

1.1 - SMBv1 und der EternalBlue-Exploit .. 2

1.2 - Der WannaCry Ransomwurm 4

2 – Angegriffene Linux-Software 5

2.1 - SSH und libSSH 5

2.2 - OpenSSL und Hearbleed 6

3 – Prävention 7

3.1 - Prävention als End-Nutzer 7

3.2 - Prävention als Entwickler 10

Fazit 11

Quellenverzeichnis 12

Angriffsszenarien auf etablierte Netzwerkprotokolle

In unserer heutigen Zeit baut die tagtägliche Nutzung von *Smartphones* und *Computern*, für das *Surfen* im *Internet* auf einige Protokolle auf. Diese definieren Schnittstellen und Methoden, um mit anderen Protokollen interagieren zu können. Allerdings treten hierbei durch das Mensch-sein der Entwickler Fehler auf, die von anderen ausgenutzt werden können. Im folgenden werden drei Beispiele von sogenannten Exploits veranschaulicht und es werden Empfehlungen aufgestellt, um die Sicherheit, des Otto Normalverbrauchers und der Protokolle die man Schreibt, im Netz zu verbessern.

1: Angegriffene Windows-Software

1-1: SMBv1 und der EternalBlue-Exploit

Im März 2017 legte Microsoft, Hersteller des weitverbreiteten Betriebssystems Windows, mit dem `Security Bulletin MS17_010`[1] eine neue Version des Betriebssystems bereit, um angreifbare Windows-Systeme gegen den sogenannten "*EternalBlue-Exploit*" zu schützen. Dieser Exploit ermöglicht es einem Angreifer, Befehle mit den selben Rechten wie das Betriebssystem selbst, auszuführen.

Das Server Message Block (= SMB) Protokol, wird genutzt um Daten zwischen zwei Rechnern im selben Netzwerk auszutauschen und wurde von IBM für diesen Zweck entwickelt.

EternalBlue baut auf drei verheerende Fehler in der Windows-Implementation des Protokolls auf. [2]

1. Die Windows-Implementation des Protokolls definiert eine Funktion, namens `srv!SrvOS2FeaListSizeToNt`, die versucht eine Listenstruktur von Metadaten vom `OS2 FEA` zum `NT FEA` Typen umzuwandeln.

Allerdings kommt es in der Funktion zu einem *Integer Underflow*, weswegen weniger Speicherplatz zugeteilt wird, als eigentlich nötig wäre. Dies führt später zu einem *BufferOverflow*.

2. Dem Zielsystem werden nun zwei Befehle gesendet, die diesem mitteilen sollen, dass Daten gesendet werden.

Der erste Befehl, `SMB_COM_NT_TRANSACT` (= `"NT_TRANSACT"`), lässt doppelt soviel Speicherplatz bereitstellen wie der zweite Befehl, `SMB_COM_TRANSACTION2` (= `"TRANSACTION2"`).

Da die Befehle sehr schnell hintereinander gesendet werden, wird nur der zweite, also `TRANSACTION2`, ausgeführt, wodurch im Zielsystem zu wenig Speicherplatz bereitgestellt wird.

3. Nach dem erreichten *BufferOverflow* ist es durch ein Verfahren namens *heap spraying* möglich, an einer beliebigen, den Angreifern bekannten Speicherstelle Speicherplatz bereitzustellen. Durch das Überschreiben der Daten an der bereitgestellten Speicherstelle mit `Batch`-befehlen, ist es möglich diese Befehle mit den selben Rechten auszuführen, wie das Betriebssystem selbst, da der SMB-Server mit Systemrechten ausgeführt wird.

1-2: Der WannaCry Ransomworm

Zwei Monate nachdem Microsoft das Security Bulletin *MS17_010* herausgegeben hat, hat sich ein Ransomworm verbreitet, welcher wie einige andere Viren unter einigen Namen Bekannt geworden ist, unter anderem als: *"WannaCry"*, *"WannaCrypt0r"* und *"WannaDecrypt0r"*. Im folgenden wird der Ransomworm des Verständnisses halber als *"WannaCry"* betitelt.

Der Wurm nutzt unter anderem als V*erbreitungsvektor* den *EternalBlue-exploit*, durch diesen kann auf dem Zielsystem das sogenannte *Payload* installiert werden. Im Fall von WannaCry wird neben dem Virus selbst eine sogenannte *Backdoor*, Namens *DoublePulsar*, installiert, mit der der Wurm später besser auf das System zugreifen kann, *DoublePulsar* ermöglicht es einem Angreifer auch das infizierte System fernzusteuern.[1]

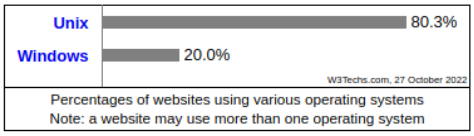
Während WannaCry versucht sich zu verbreiten, verschlüsselt es auch die Dateien des Zielsystems. Für diesen Zweck wird ein sogenanntes *Schlüsselpaar* und ein *AES-Schlüssel* pro Installation generiert. Aufgrund dessen, das das Verschlüsseln mit AES ein sehr schneller Vorgang ist, kann das gesamte System innerhalb von Minuten verschlüsselt werden.

Daraufhin wird der AES-Schlüssel mit dem öffentlichen Schlüssel verschlüsselt und separat gespeichert. Der dazugehörige private Schlüssel wird daraufhin mit dem öffentlichen Schlüssel des Kontrollservers verschlüsselt.

Das bedeutet, dass WannaCry nur den geheimen Schlüssel zur Geisel nimmt, und verspricht diesen nach erfüllter Zahlung freizugeben.[2] Selbst nach einer erfüllten Zahlung ist es möglich, das die Angreifer den Schlüssel nicht Freigeben, weswegen unter anderem das Verbraucherzentrum dagegen rät, das Lösegeld zu zahlen. [3]

2: Angegriffene Linux-Software

Auch Systeme die nicht auf Windows beruhen, wurden und werden immernoch angegriffen.



Unter den 10 Millionen größten Webseiten, nutzen ca. 80% ein Betriebssystem, welches zur Linux-Betriebssystemfamilie gehört[1].

2-1: SSH und libSSH

`libSSH` ist eine Bibliothek, die für die Programmiersprachen *C* und *C++* entwickelt wurde, die es einem Programm erleichtert das SSH (*Secure Shell Protocol*) zu nutzen.

Meistens nutzt man SSH um andere Server oder Rechner fernzusteuern. Dafür muss man sich als ein, dem Server bekannter, Benutzer authentifizieren und bekommt damit die selben Rechte zugewiesen, die der Nutzer an dem Rechner hat.

In bestimmten Versionen von libSSH war es möglich, diese Authentifizierung im Zielsystem zu umgehen, indem man dem Server statt der erwarteten Nachricht, `SSH2_MSG_USERAUTH_REQUEST`, die Nachricht `SSH_MSG_USERAUTH_SUCCESS`, sendete. Somit konnte ein Angreifer sich bei einem solchen angreifbarem System als `root`, also als Administrator mit allen Rechten anmelden, und das System vollständig übernehmen.[1]

Zum verschlüsselten Übertagen von Nachrichten nutzen einige Implementationen des SSH-Protokolls ein Protokoll, welches Secure Socket Layer Protokoll (= SSL) genannt wird, dieses ist auch unter dem Namen TLS bekannt geworden. Eine der am weitesten verbreiteten C/C++-Bibiliotheken für die Implementation dieses Protokolls hat allerdings einen lange Übersehenen, großen Schwachpunkt.

2-2: openSSL und Heartbleed

Apache HTTPD und NGIИX[1] zählen zu den am weitesten verbreiteten Web-Engines für Linux, diese unterstützen auch das HTTPS-Protokoll, und nutzen für die darin benötigte SSL-Implementation die *OpenSSL*-Library. Da ein Großteil der Server dieser Welt auf diesen Programmen aufbaut, haben selbst kleine Fehler große Auswirkungen, welches am folgenden Beispiel besonders gut veranschaulicht wird.

Das RFC-6520 definiert die *Heartbeat-Erweiterung* für das TLS-Protokoll, die in SSL übernommen wurde. Diese Erweiterung war dafür gedacht, die Verbindung zu einem Server zu Überprüfen.

In dieser *[sendet] der Client [...] periodisch in einem Datenfeld einen zufällig generierten String und in einem weiteren Feld dessen Länge an den Server; dieser schickt dann den String zurück an den Client. Damit kann der Client einfach überprüfen, ob der Server noch erreichbar ist.*(H1). Ein weiterer Vorteil dieser Funktion ist es, das NAT-Router und Firewalls, aufgrund der periodischen Datenübermittlung, die Verbindung nicht beenden(vgl. H1).

In OpenSSL wurde diese Erweiterung implementiert, allerdings mit einem Fehler. Dieser ermöglichte es einem die eigentliche Länge, die vom Anfang der Zeichenkette bis zum nächsten `null-terminator (\u0000)` geht, mit der bereitgestellten Länge zu Überschreiben,

Die Zeichenkette, die im Missbrauchsfall vom Server zurück gesendet wird, kann unter anderem *Passwörter*, *Benutzerinformationen*, *private RSA-Schlüssel*, und andere Informationen beinhalten, die geheim bleiben sollen. Wenn ein Angreifer es schaffen sollte, an den geheimen RSA-Schlüssel des Servers zu kommen, kann er sich unter anderem als der Server oder sich als ein anderer Benutzer ausgeben.

Kurz nach der Veröffentlichung von OpenSSL Version 1.0.1g, in welcher dieser Fehler behoben wurde, wurde es empfohlen, die Schlüsselpaare neu zu erstellen und zu zertifizieren. [2] [3] [4]

3: Prävention

Um sich gegen *Exploits* und *Vunerabilities* zu schützen gibt es mehrere Ansätze, hier werden Ansätze aus der Sicht eines Benutzers und eines Entwicklers betrachtet.

3-1: Prävention als End-Nutzer

Als End-Nutzer gibt es ein paar Details, worauf man beim Installieren von Software aufpassen sollte, um sich abzusichern:

- möglichst aus vertrauten Quellen (wie AppStores) installieren
- **immer** vor der Weitergabe identifizierender Informationen die URL überprüfen, um *Phishing* zu verhindern

Allerdings kann man auch als End-Nutzer aktiv dabei helfen, die Programme sicherer zu gestalten. Einige Apps und Services bieten sogenannte *Bug-bounties* an, bei denen man Geld o.ä. bekommt, wenn man einen *Bug* in der Software nachweist.

Bei Apps und Services, die so etwas nicht anbieten ist man *moralisch*, aber *nicht rechtlich*, dazu verpflichtet, diese den Entwicklern verantwortlich mitzuteilen, dies nennt man ***Responsible Disclosure***.

Responsible Disclosure geschieht meist in einer ähnlichen Reihenfolge wie die, die hier angegeben ist:

1. Privaten Kontakt – z.B. über E-Mails – suchen
2. Erklärung (und/oder Demonstration) der Schwachstelle
3. Warten auf Aktionen der Herstellerseite
    - Die Zeitspanne liegt hier meist zwischen circa einer Woche bis circa zwei Monate
4. Wenn nach einer längeren Zeit keine Aktionen von Herstellerseite (wie z.B. ein *Bugfix* oder *Update*) bemerkbar sind, oder wenn der *Bugfix* bereits erschienen ist, *kann* man auf die Schwachstelle öffentlich hinweisen
    - Meist wird dieser Schritt dazu genutzt, um die öffentlichkeit auf das Problem aufmerksam zu machen und mehr Nutzer dazu zu ermuntern, das neue *Update* herunterzuladen.

Nunmal ist die ganze Prozedur, nach Schwachstellen zu suchen, eine Art rechtliche Grauzone, in der es zwar nicht illegal ist, eine Schwachstelle zu finden und auszunutzen, *solange* man nicht die Daten anderer ohne deren Einwilligung erlangt. Um die Existenz der Schwachstelle zu Beweisen, nutzen die meisten daher eigene, seperate Konten. Wenn man sich nicht sicher ist, ob dies sowohl rechtlich als auch von den AGBs erlaubt ist, kann man sich dafür einen Anwalt zu Rat ziehen, der diese für einen durchliest.[1]

Als End-Nutzer ist eine Benachrichtigung über ein *Update* etwas, was man vermeiden möchte, und das *Update* selber, wird von den meisten auch immer solange wie möglich gemieden. Jedoch so lästig wie sie denn nun mal sein mögen sind *Updates* ein wichtiger Faktor der Prävention von Schwachstellen auf der Seite der End-Nutzer. Einige *Updates* bestehen fast nur aus *Bugfixes* und helfen damit das Produkt vor Angreifern zu schützen, indem sie die Schwachstellen beheben.

Um diese Barriere, die die Nutzer im Kopf haben, zu umgehen gibt es einige Ansätze, unter anderem verwendet *Windows 10* einen Ansatz, welcher die Nutzer fast schon dazu zwingt, das neue *Update* zu installieren. Dies hilft zum Beispiel unerfahrenen Benutzern, das eigene System auf dem neuesten Stand zu halten.

Ein anderer Ansatz wird von Betriebssystemen wie *Arch Linux* genutzt. Diese sogenannten "*rolling-release*" Betriebssysteme haben sehr oft sehr kleine *Updates*, die meist im Hintergrund automatisiert installiert werden. Bei solchen Betriebssystemen wird es häufig empfohlen, meistens mindestens einmal die Woche, diese Updates herunterzuladen, um das System auf dem neuesten Stand zu halten.

Eine weitere Option als End-Nutzer ist es, *Open-Source* Software zu bevorzugen, da diese meistens aufgrund des öffentlich zugänglichen Quelltextes ein paar Vorteile mit sich bringen.

Das Benutzen aktueller Programmiersprachen und -standards
Veranlasst durch einen Mangel an Programmierern, die Programme aus alten bzw. unsicheren Programmiersprachen in neuere Programmiersprachen übersetzen können, gibt es noch Programme die auf Sprachen wie *COBOL*[2](#3-12-cobol-annotation) laufen und einige Schwachstellen beinhalten können. Dies kann man in sogenannten Closed-Source Programmen relativ gut verstecken, was bei Programmmen mit öffentlichem Quelltext nicht funktioniert. Hier wird

man durch eine Art Gruppenzwang dazu aufgefordert mit erhöhtem Standard zu Programmieren. Dies kann zur Sicherheit der Software beitragen.

Das bessere Benutzen von Kommentaren
Durch den Gruppenzwang wird man auch dazu aufgefordert den eigenen Quellcode zu Dokumentieren, so dass andere ihn leichter Verstehen können. Dies führt dazu, das der Code besser beschrieben ist, wodurch das *Debugging* einfacher wird, welches auch zur Sicherheit der Software beiträgt.

Allerdings ist selbst Open-Source Software nicht immer fehlerlos und kann gravierende Fehler beinhalten (siehe Kapitel [2-2]), die relativ einfach übersehen werden können.

3-2: Prävention als Entwickler

Als Entwickler ist man dafür verantwortlich, den eigenen Quellcode so zu gestalten, dass er in möglichst allen Fällen so reagiert wie gewünscht, da Entwickler allerdings auch nur Menschen sind, ist es nunmal möglich, dass ein unwahrscheinlicher Fall ausgelassen wird, wodurch Sicherheitslücken entstehen können.

Um Sicherheitslücken zu verhindern, nutzen die meisten Firmen sogenannte *Code Reviews*, die aufgrund des geringeren Zeit- und Ressourcenverbrauchs automatisiert durchgeführt werden.

Für Entwickler gibt es mehrere Möglichkeiten, den eigenen Quelltext dazu zu bringen, so zu reagieren, wie er soll. Hier sind ein paar Methoden, die dafür genutzt werden:

Kommentare

Kommentare sind eines der effektivsten Mittel, die in so gut wie allen Programmiersprachen von Anfang an eingebaut sind. Sie helfen dabei, den eigenen Quellcode leserlicher zu gestalten und diesen für die bessere Zusammenarbeit mit Schnittstellen und anderen Entwicklern zu dokumentieren, so dass andere und man selbst, auch nach mehreren Monaten oder Jahren nachvollziehen kann, was der eigene Quelltext bewirken soll.
In dieser Hinsicht ist er ähnlich einer Handschrift, die man selbst schnell und einfach lesen kann, während andere zum "entziffern" länger brauchen.

Rubber Duck Debugging

So lächerlich sich der Name der Debuggingmethode auch anhört, so effektiv ist sie auch. In dieser Debuggingmethode wird der eigene Quellcode Zeile für Zeile einer Gummiente, einem Haustier oder wahlweise Mitarbeiter erklärt, bis zu der Zeile, in welcher sich der Fehler im Quelltext befindet.
Diese Methode ist eine, die einen sehr großen Anklang unter Entwicklern gefunden hat, neben Code Tests/Reviews, Logging und `printf()` statements. [1]

Version-Control-Systems (VCS)

Eines der am weitesten verbreiteten Werkzeuge beim entwickeln sind sogenannte Versionskontrollsysteme, wovon eines der bekanntetesten *git* ist. Diese helfen dabei, nachzuvollziehen wann und wo eine Funktion oder eine Schwachstelle zum Programm hinzugefügt wurde. Wenn man dann zum Beispiel git mit einem externen *Host* wie *GitHub.com* oder dem sogenannten *Arch User Repository (AUR)* benutzt, kann man den eigenen Code sowohl verbreiten, als auch diese Webseiten als externes Backup nutzen, für den garnicht mal *so* unwahrscheinlichen Fall, dass die Projektdateien auf der eigenen Festplatte irreparabel beschädigt sind.

Fazit

Abschließend kann man sagen, dass die Sicherheit und Integrität der Software stark von der Qualität des Quellcodes abhängt. Diese ist meist in *Open-Source* Software etwas höher als in *Closed-Source* Software. Allerdings haben die meisten Programme unabhängig von der Einsichtbarkeit des Quellcodes Fehler, die man ausnutzen kann, da die Entwickler auch nur Menschen sind.

Gute Mittel zur Fehler-Prophylaxe auf Seiten der Entwickler sind **Kommentare**, **Versionskontrollsysteme** und **Code-Reviews**. Gute Mittel um sich als Otto Normalverbraucher gegen *Phishing* und *Exploits* zu schützen sind *das Überprüfen der URL vor der Weitergabe persönlicher Informationen* und das *Installieren von Programmen aus vertrauenswürdigen Quellen* wie unter anderem den vorinstallierten: *"Google Play Store", "Apple Store", "Snap Store", "Flatpak"* oder *"Microsoft Stores"*.

Quellenverzeichnis

> Der Name wird gebildet aus: Kapitelname/Quellennr
> Innerhalb des Kapitels sind diese nur als [Quellennr] angegeben.
> z.B. ist `1-1/1` die erste Quelle des Kapitels 1-1

H1: Exploit

→ Seite 486

```
Voller Titel:   Exploit! Code Härten, Bugs analysieren,
Hacks verstehen
     ISBN-13:   978-3-83626598-0
      Verlag:   Rheinwerk Computing, Bonn
     Auflage:   1. Auflage 2019

Authoren:
  - Dr. Klaus Gebeshuber
  - Dr. Egon Teinik
  - Dr. Wilhelm Zugaj
```

1-1/1: MS17_010

Quelle: Microsoft;
· MS17_010 ist auffindbar unter https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2017/ms17_010, (Englisch);
Abgerufen am 8. August 2022

1-1/2: SentinelOne

Quelle: SentinelOne;
· Auffindbar unter https://sentinelone.com/blog/eternalblue-nsa-developed-exploit-just-wont-die, (Englisch);
Abgerufen am 12. August 2022

1-2/1: Kaspersky

Quelle: Kaspersky Antivirus
· Auffindbar unter https://kaspersky.com/resource-center/threats/ransomware-wannacry, (Englisch);
Abgerufen am 8. August 2022

1-2/2: Computerphile

Quelle: Youtube
· Auffindbar unter https://youtube.be/pLluFxHrc30, (Englisch);
Abgerufen am 8. August 2022

> Michael "Mike" Pound ist ein Forscher an der Universität Nottingham, der sich unter anderem mit dem Themenkomplex *computer security* befasst.

1-2/3: Zahlen?

Quellen: Verbraucherzentrale, BSI, Microsoft, nomoreransom.org
· Auffindbar unter:
· Verbraucherzentrale:
https://www.verbraucherzentrale.de/wissen/digitale-welt/phishingradar/was-ist-ransomware-und-wie-kann-ich-mich-schuetzen-69789#3
· BSI: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Cyber-Sicherheitslage/Methoden-der-Cyber-Kriminalitaet/Schadprogramme/Ransomware/ransomware_node.html
· Microsoft: https://support.microsoft.com/de-de/windows/sch%C3%BCtzen-ihres-pc-vor-ransomware-08ed68a7-939f-726c-7e84-a72ba92c01c3
· nomoreransom.org:
https://www.nomoreransom.org/en/ransomware-qa.html#payment, (Englisch);
Abgerufen am 19. Oktober 2022

> **nomoreransom.org** ist eine Initiative der *Nationalen High Tech Crime Unit der **Niederländischen Polizei**, des Zentrums für Europäische Cyberdelikte der **Europol**, Kaspersky*, und *McAfee*, die es sich zum ziel gesetzt hat, den Opfern von Ransomware dabei zu helfen, ihre Dateien zurückzubekommen, ohne das Lösegeld zu zahlen.

2-0/1: Websurvey

Quelle: W3Techs
· Auffindbar unter:
https://w3techs.com/technologies/overview/operating_system,
(Englisch);
Abgerufen am 27. Oktober 2022

> Bild: © 2022 W3Techs, Zurechtgeschnittener Screenshot

2-1/1: libSSH Statement

Quelle: libSSH.org
· Auffindbar unter: https://www.libssh.org/security/advisories/CVE-2018-10933.txt, (Englisch);
Abgerufen am 16. August 2022

2-2/1: nginx Annotation

> Aussprachehilfe: /ɛnd͡ʒɪnks/, benannt nach dem Englischen:
> "Engine X"

2-2/2: OpenSSL

Quelle: OpenSSL.org
· Auffindbar unter: https://www.openssl.org/news/secadv/20140407.txt,
(Englisch);
Abgerufen am 18. August 2022

2-2/3: Heartbleed

Quelle: Heartbleed.com
· Auffindbar unter: https://heartbleed.com, (Englisch);
Abgerufen am: 18. August 2022

2-2/4: Cisa

Quelle: US Cybersecurity and Infrastructure Security Agency (CISA)
NCAS Alert TA14-098A
· Auffindbar unter: https://www.cisa.gov/uscert/ncas/alerts/TA14-098A,
(Englisch);
Abgerufen am: 18. August 2022

3-1/1: Rechtliches

Quelle: Youtube/Tom Scott
· Video-Titel: `The Moonpig Bug: How 3,000,000 Customers' Details Were Exposed`
· Auffindbar unter: https://youtu.be/CgJudU_jlZ8, (Englisch);
Abgerufen am: 31. Oktober 2022

3-1/2: COBOL Annotation

COBOL (="COmmon Business Oriented Language") ist eine Programmiersprache die 1959 (13 Jahre vor der ersten Version von ***C*** in 1972) entwickelt wurde.

3-2/1: Rubber Duck debug

Quelle: RubberDuckDebugging.com
· Auffindbar unter: https://rubberduckdebugging.com, (Englisch);
Abgerufen am: 24. August 2022