Zentrum für sichere Informationstechnologie – Austria
Secure Information Technology Center – Austria

A-SIT

Seidlgasse 22 / 9, 1030 Wien
Tel.: (+43 1) 503 19 63–0
Fax: (+43 1) 503 19 63–66

Inffeldgasse 16a, 8010 Graz
Tel.: (+43 316) 873-5514
Fax: (+43 316) 873-5520

http://www.a-sit.at
E-Mail: office@a-sit.at

DVR: 1035461
ZVR: 948166612
UID: ATU60778947

IOS SCHNITTSTELLE ZUR NETZWERKVERKEHRSANALYSE

Version 1.0 vom 08.06.2020
Dominik Mocher – dominik.mocher@iaik.tugraz.at

Abstract/Zusammenfassung: Dieses Projekt untersucht drei Methoden, um Netzwerkverkehr auf einem iOS Gerät aufzuzeichnen und anschließend analysieren zu können. Im ersten Ansatz werden Methoden des Network Extension Frameworks zur Laufzeit durch eigene Implementierungen ersetzt, die übertragene Daten protokollieren sollen. Im zweiten Ansatz wird zu diesem Zweck ein Proxy-Server konfiguriert, der als Teil einer App am Gerät ausgeführt wird. Als letzte Methode wird an Stelle des Proxy-Servers ein VPN-Server genutzt.

Inhaltsverzeichnis

1. Einleitung

In diesem Projekt wurde untersucht, ob der Netzwerkverkehr eines iPhones direkt am Gerät aufgezeichnet werden kann. Die Aufzeichnung kann danach zur weiteren Analyse in beliebigen unterstützten Formaten vom iPhone exportiert werden. Es sollen für die Aufzeichnung keine erweiterten Zugriffsrechte mittels Jailbreak freigeschaltet werden müssen. Diese Einschränkung ermöglicht es, den Netzwerkverkehr von Applikationen zu analysieren, die ihre Funktion bei erweiterten Rechten verändern oder einstellen.

Dazu werden drei Ansätze auf einem iPhone mit aktuellem iOS 13.5 betrachtet. Der erste Ansatz beruht auf der Möglichkeit, die von iOS-Applikationen genutzte Objective C-Laufzeitumgebung während der Ausführung verändern zu können. Hierbei können existierende Methoden von Klassen und Objekten durch eine andere Implementierung ersetzt werden. Eine Alternative dazu ist der Betrieb eines lokalen Proxy-Servers als Teil einer iOS-App. Sollte dieser in den Einstellungen des Betriebssystems eingerichtet werden, so wird der gesamte Netzwerkverkehr der WLAN-Schnittstelle darüber gesendet.
Als dritter Ansatz wurde der Proxy-Server durch einen lokalen Virtual Private Network (VPN)-Server ersetzt.

2. Verändern der Netzwerkmethoden

Objective C referenziert Methoden von Klassen und Objekten mit Hilfe von Selektoren. Diese sind eindeutig, werden aus dem Methodennamen sowie den Methodenparametern gebildet und bleiben auch nach dem Kompilieren bestehen. Die Objective C-Laufzeit bietet jedoch die Möglichkeit, das Ziel eines Selektors während der Ausführung zu verändern, so dass auf eine andere Methode mit gleichen Parametern verwiesen wird. Diese Technik wird als „Method Swizzling“[1] bezeichnet.

Da bestehende Klassen in Objective-C mit neuen Methoden erweitert werden können, erlaubt Method Swizzling in iOS inkludierte Bibliotheken wie das Network Extension Framework[2] zu verändern. Dieses Framework ermöglicht die Konfiguration der Netzwerkschnittstellen und beinhaltet private Lese- und Schreibmethoden für das Transmission Control Protocol (TCP) und User Datagram Protocol (UDP) sowie für VPN-bezogene Klassen. Die relevanten Selektoren für diese Methoden sind nachfolgend aufgelistet.

```
NEAppProxyTCPFlow::readData:completionHandler
NEAppProxyUDPFlow::readDatagrams:completionHandler
NEPacketTunnelFlow::readPackets:completionHandler
NEPacketTunnelFlow::readPacketObjects:completionHandler
NWTCPConnection::readLength:_:completionHandler
NWTCPConnection::readMinimumLength:_:maximumLength:completionHandler

NEAppProxyTCPFlow::write:_:withCompletionHandler
NEAppProxyUDPFlow::writeDatagrams_:sentBy:completionHandler
NEPacketTunnelFlow::writePackets:_:withProtocols
NEPacketTunnelFlow::writePacketObjects:
NWTCPConnection::write:_:completionHandler
NWUDPSession::writeMultipleDatagrams:_:completionHandler
NWUDPSession::writeDatagram:_:completionHandler
```

Mittels Method Swizzling wurde die Implementierung der angeführten Methoden ausgetauscht. Im Falle von lesenden Methoden wird der Selektor auf eine angepasste Implementierung geändert, die den originalen Code aufruft, die zurückgegebenen Daten aufzeichnet und erst danach an die aufrufende Methode retourniert. Bei Methoden mit Schreibzugriff werden die als Parameter übergebenen Daten zuerst aufgezeichnet und anschließend an die ursprüngliche Implementierung weitergegeben. Die Anpassungen wurden in der Open Source App Lockdown[3] durchgeführt, da diese bereits einen systemweiten VPN-Client auf Basis des Network Extension Frameworks einrichtet und startet.

Trotz der beschriebenen Änderungen konnten mit diesem Ansatz keine Pakete aufgezeichnet werden, da die veränderten Methoden bei aktiver VPN-Verbindung nicht aufgerufen wurden. iOS scheint zu diesem Zwecke eine eigene Objective C-Laufzeit zu verwenden, die Teil des Betriebssystems ist. Da jede App eines iPhones ohne Jailbreak in einer eigenen Sandbox[4] ausgeführt wird, kann die Applikation nicht auf Daten und Schnittstellen außerhalb zugreifen. Dadurch können die entsprechenden Methoden in der Laufzeit des Betriebssystems nicht von der App aus verändert werden.

[1] https://nshipster.com/method-swizzling/
[2] https://developer.apple.com/documentation/networkextension
[3] https://github.com/confirmedcode/lockdown-ios
[4] https://developer.apple.com/app-sandboxing/

3. Lokaler Proxy-Server

Dieser Ansatz beruht auf der Idee, einen Hypertext Transfer Protocol (HTTP)-Proxy Server als App am iPhone auszuführen. Die Implementation des Proxy-Servers in der Demo-App beruht auf TCP Sockets der CocoaAsyncSockets-Bibliothek[5], welche die von iOS verwendeten BSD-Sockets[6] abstrahiert. Wenn eine Applikation eine HTTP-Nachricht an einen Server schickt, wird diese zuerst an einen Socket des Proxy-Servers übertragen. Dort kann die Nachricht gelesen und aufgezeichnet werden, bevor sie an die eigentliche Destination weitergeleitet wird. Die Antwort auf die ursprüngliche Nachricht wird ebenfalls zuerst an den Proxy-Server geschickt, der diese wiederum aufzeichnen kann und anschließend weiterleitet.

Der Proxy-Server muss manuell in den Einstellungen für die WLAN-Schnittstelle konfiguriert werden und zeichnet erst danach sämtliche HTTP-Nachrichten, die über diese Schnittstelle gesendet werden, auf. iOS-Applikationen müssen aufgrund der Development Guidelines von Apple[7] zwingend das Hypertext Transfer Protocol Secure (HTTPS) verwenden. Dabei werden die Nachrichten verschlüsselt und können daher nur in dieser Form aufgezeichnet werden. Für den praktischen Einsatz ist es daher notwendig, den Proxy-Server um HTTPS-Funktionalität zu erweitern. Um die Nachrichten anschließend unverschlüsselt aufzuzeichnen, muss dieser ein eigenes Server-Zertifikat anbieten. Das Zertifikat muss vom Benutzer oder der Benutzerin manuell in die Keychain des Systems importiert werden und anschließend als vertrauenswürdig eingestuft werden.

Trotz der möglichen Integration von HTTPS-Funktionalität bleiben mehrere Limitierungen bestehen. Sollten Apps etwa den Server mittels Certificate Pinning[8] validieren, so würde statt des erwarteten Zertifikats des Zielservers das Zertifikat des Proxy-Servers präsentiert und die Verbindung verweigert werden.

Da er sich um einen reinen HTTP-Proxy handelt, kann keine Information auf Ebene von TCP, UDP oder des Internet Protocols (IP) gesammelt werden. iOS unterstützt darüber hinaus nur die Verwendung eines Proxy-Servers für Netzwerkverbindungen der WLAN-Schnittstelle. Daten, die über das Mobilfunknetz gesendet werden, werden nicht über den Proxy-Server geleitet und können daher auch nicht aufgezeichnet werden.

Ein weiteres Problem ist die Verwaltung von Hintergrundaktivitäten durch das Betriebssystem. Rechenintensive Programme können beim Verschieben in den Hintergrund jederzeit unterbrochen werden, was die weitere Aufzeichnung von HTTP-Nachrichten verhindert. Wenn der Proxy-Server als aktive App im Vordergrund bleibt, kann diese Einschränkung zwar umgangen werden, resultiert dabei aber in einer starken Erwärmung des iPhones.

4. Lokaler VPN-Server

In diesem Ansatz wird an Stelle eines HTTP-Proxy-Servers ein VPN-Service direkt am iPhone betrieben. Mit Hilfe des bereits erwähnten Network Extension Frameworks kann der VPN-Client in den Systemeinstellungen von der App aus eingerichtet und aktiviert werden. In der Demo-App wurde L2TP/IPsec [1] als VPN-Protokoll verwendet, es können aber auch Cisco IPsec[9] oder IKEv2 [2] verwendet werden.

Wie bereits zuvor wird die CocoaAsyncSockets-Bibliothek für die Implementierung der in L2TP/IPSec erforderlichen UDP-Sockets verwendet. Bei Netzwerkanfragen einer App werden bei aktivem VPN-Client die entsprechenden IP-Pakete vom Client in UDP-Datagrams gepackt und an

[5] https://github.com/robbiehanson/CocoaAsyncSocket/
[6] https://github.com/apple/darwin-xnu/blob/master/bsd/sys/socket.h
[7] https://developer.apple.com/documentation/security/preventing_insecure_network_connections
[8] https://developer.apple.com/documentation/foundation/url_loading_system/handling_an_authentication_challenge/performing_manual_server_trust_authentication
[9] https://www.cisco.com/c/en/us/td/docs/net_mgmt/vpn_solutions_center/2-0/ip_security/provisioning/guide/IPsecPG1.html

den Socket des VPN-Servers der Demo-App geschickt. Der lokale VPN-Server zeichnet das entpackte IP-Paket auf und leitet es an die korrekte Destination weiter. Für das Ziel ist der VPN-Server der Ursprung der Verbindung und bekommt daher auch die zur Antwort gehörenden Pakete. Erneut werden diese aufgezeichnet und nach dem Verpacken in UDP-Datagrams an die ursprüngliche App geleitet.

Durch dieses Vorgehen ist es im Gegensatz zum zuvor beschriebenen Proxy-Server möglich, Daten der Protokolle TCP, UDP und IP auszulesen und zu speichern. Der Implementierungsaufwand ist aufgrund der Komplexität der VPN-Protokolle höher, weshalb in der Demo-App kein vollständiger VPN-Server umgesetzt wurde.

Wie schon im Ansatz zuvor wird der Server durch die Prozessverwaltung von iOS gestoppt, sobald die Applikation nicht mehr im Vordergrund ausgeführt wird. Diese Einschränkung kann auf dem gleichen Wege umgangen werden, führt aber wie schon beim Proxy-Server zu starker Wärmeentwicklung.

5. Conclusio

In diesem Projekt wurden drei Ansätze zum Aufzeichnen von Netzwerkverkehr auf iOS-Geräten untersucht. Das Ändern der Implementierungen von Lese- und Schreibmethoden aus dem iOS Network Extension Framework erwies sich dabei als nicht zielführend. Die entsprechenden Methoden konnten zwar in der Laufzeitumgebung der Applikation ausgetauscht werden, beeinflussten jedoch nicht die vom Betriebssystem genutzte Implementation.

Die Verwendung eines lokalen Proxy-Servers ermöglicht das Auslesen von HTTP- und HTTPS-Nachrichten am Gerät, das Aufzeichnen ist jedoch auf die WLAN-Schnittstelle beschränkt und kann für Netzwerkprotokolle wie IP, TCP und UDP auf diesem Wege nicht durchgeführt werden.

Diese Pakete können durch den Einsatz eines VPN-Servers, der als Teil einer App am Smartphone ausgeführt wird, protokolliert werden. Zudem ist dieser nicht auf eine Netzwerkschnittstelle beschränkt und erfasst so auch Pakete, die mittels Mobilfunknetz übertragen werden. Da die einzige Limitierung des lokalen VPN-Servers durch das Prozessmanagement von iOS verursacht wird und mit einer Ausnahmegenehmigung von Apple beseitigt werden könnte, stellt dieser den vielversprechendsten Ansatz zum Protokollieren von Netzwerkverkehr am Gerät dar. Die dadurch gewonnenen Aufzeichnungen können anschließend für die Analyse des Netzwerkverkehrs von Apps herangezogen werden.

Referenzen

[1] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn und B. Palter, „RFC2661: Layer Two Tunneling Protocol,“ 1999.

[2] C. Kaufman, P. Hoffman, Y. Nir und P. Eronen, „RFC7296: Internet Key Exchange Protocol Version 2,“ 2014.