MONTHLY SECURITY SUMMARY

)SCIP(

AUSGABE APRIL 2022
REQUEST SMUGGLING UND HONEYTOKENS

REQUEST SMUGGLING BEIM TESTEN NUTZEN

Durch Request Smuggling lassen sich HTTP-Kommunikationen beeinflussen. Wie diese Technik im Rahmen von Sicherheitstests eingesetzt werden kann, erklärt unser Artikel.

HONEYTOKENS GEGEN ANGREIFER EINSETZEN

Das Prinzip von Honeytoken ähnelt Minesweeper, denn Angreifer werden mit der Interaktion mit diesen als solche auffallen. Wir erklären Ihnen, wie Sie das zu Ihrem Vorteil einsetzen können.

April 2022: Der NATO-Cyber-Bündnisfall

Aggressive Aktivitäten im Cyberspace können einen *NATO-Bündnisfall* auslösen. So simpel dieser Grundsatz klingen kann, so weit ist seine vermeintliche Einfachheit von der nicht-digitalen Realität entfernt. Digitale Angriffe geographisch zurückzuverfolgen ist sehr *schwierig*. Die meisten Angreifer, egal ob mit wirtschaftlichen oder politischen Interessen, werden ihre Spuren verschleiern wollen. Dadurch kann eine Rückverfolgung erschwert und sich damit einem Zugriff entzogen werden.

Als Angriffsziel ist es in den allermeisten Fällen deshalb unmöglich zu bestimmen, wo der Angriff herkam. Die identifizierte Quelle könnte *gefälscht* sein. Es gibt genügend Literatur dazu, wie dies mittels *Proxies* und *IP-Spoofing* getan werden kann.

Erst die Zusammenarbeit mit Providern macht es möglich, die wahren Urheber zu identifizieren. Die bewusste *Internationalisierung* von Angriffen führt dazu, dass dann mit Behörden aus verschiedenen Ländern zusammengearbeitet werden muss. Ein sehr langwieriger und mühsamer Prozess. Nur die wenigsten Opfer wollen diese Aufwände auf sich nehmen. Man konzentriert sich lieber auf die Abwehr zukünftiger Angriffe, als die Nachwehen der vergangenen Niederlagen aufrecht zu erhalten.

Wenn also mal wieder verkündet wird, dass die *Russen* oder *Chinesen* für einen Zwischenfall verantwortlich sind: Seien Sie skeptisch! In diesen Regionen gibt es eine Vielzahl ungesicherter Systeme, die sich kapern und als Zwischenstationen für Angriffe missbrauchen lassen. Und die für die Aufklärung erforderliche Kooperation der Behörden vor Ort lässt stark zu wünschen übrig.

Marc Ruef
Head of Research

NEWS

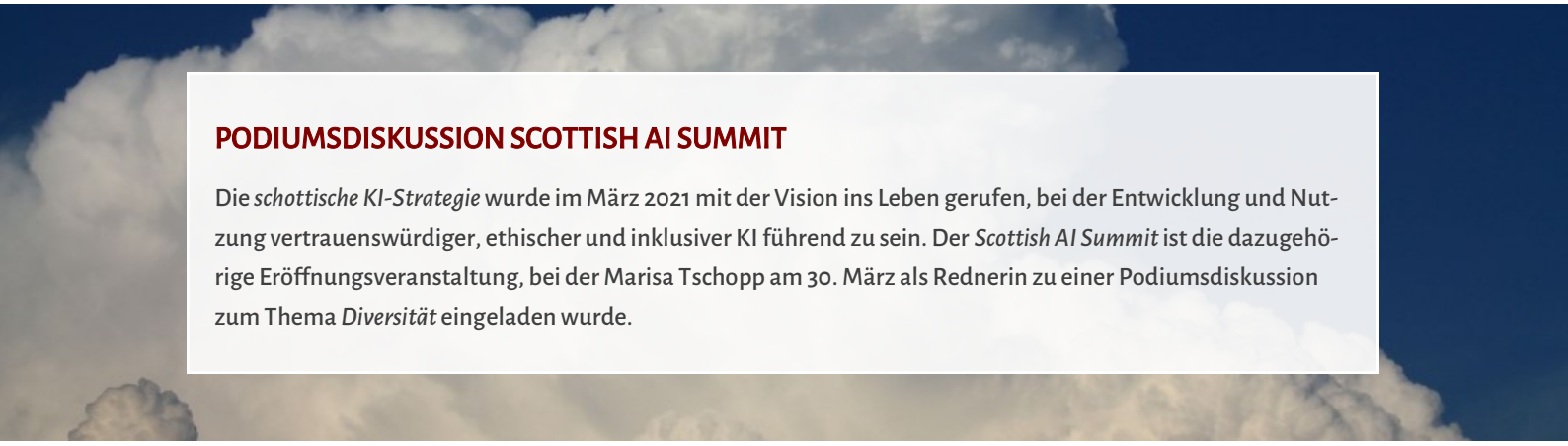
WAS IST BEI UNS PASSIERT?

INTERVIEW IN CYBERNEWS MIT CEO TOMASO VASELLA

Das Magazin *CyberNews* beschäftigt sich mit Themen rund um *Cybersecurity* und *neue Technologien*. In der aktuellen Ausgabe hat die Redakteurin Anna Zhadan ein umfangreiches Interview mit Tomaso Vasella, CEO der scip AG, geführt. Darin geht es einerseits um die *strategische Ausrichtung* des Unternehmens. Aber auch um generelle Aspekte, die die *Cybersecurity-Industrie* geprägt haben und zukünftig beeinflussen werden.

ANALYSE ZU VERMEINTLICH FABRIZIERTEM VIDEO VON WOLODYMYR SELENSKYJ

Die Zeitung *TagesAnzeiger* berichtet über Aktivitäten und Hintergründe des *Konflikts in der Ukraine*. Eine der Ansprachen des ukrainischen Präsidenten *Wolodymyr Selenskyj* stand dabei zur Diskussion. Michèle Trebo und Marc Ruef haben eine technische Analyse des Videos vorgenommen, um die Echtheit dessen zu bestimmen. Es verschiedene Hinweise, die auf einen *Greenscreens* oder eine anderweitige *Komposition* hindeuten.



PODIUMSDISKUSSION SCOTTISH AI SUMMIT

Die *schottische KI-Strategie* wurde im März 2021 mit der Vision ins Leben gerufen, bei der Entwicklung und Nutzung vertrauenswürdiger, ethischer und inklusiver KI führend zu sein. Der *Scottish AI Summit* ist die dazugehörige Eröffnungsveranstaltung, bei der Marisa Tschopp am 30. März als Rednerin zu einer Podiumsdiskussion zum Thema *Diversität* eingeladen wurde.

Weitere News zu unserem Unternehmen finden Sie auf unserer Webseite.

SCIP BUCHREIHE

UNSER AKTUELLES JAHRBUCH

Erneut veröffentlichen wir die aktuelle Ausgabe unseres Jahrbuchs. Bereits zum 11ten Mal fassen wir in diesem die Fachbeiträge von einem Jahr Forschung im Bereich Cybersecurity zusammen.

Das Buch ist wiederum sowohl in deutscher (ISBN 978-3-907109-26-7) als auch in englischer Sprache (ISBN 978-3-907109-27-4) verfügbar. Das Vorwort wurde von Marko Rogge, seines Zeichens IT-Sicherheitsbeauftragter für Mobile Security bei der Deutschen Bahn AG, verfasst.

In unserem Katalog finden sich ebenfalls themenspezifische Bücher zu Künstlicher Intelligenz (ISBN 978-3-907109-14-4) und Sicherheit in der Hotellerie (978-3-907109-16-8).

Weitere Informationen und Bestellungen auf unserer Webseite.

ISBN 978-3-907109-26-7 [de]
ISBN 978-3-907109-27-4 [en]

SICHERHEIT IST KEINE GLÜCKSSACHE

Bildquelle: https://pixabay.com/images/id-721951/

ANDREA HAUSER

REQUEST SMUGGLING BEI EINEM SECURITY TEST ANWENDEN

Bei *Request Smuggling* handelt es sich um eine Schwachstelle, die dann auftritt, wenn mehrere Server an der Bearbeitung einer Anfrage beteiligt sind und sich diese Server nicht einig sind, wo der Body der Anfrage beginnt und endet. Beispiele für solche Ketten von Servern sind Reverse-Proxys mit ihrem Back-End oder das vielfach eingesetzte Load-Balancing, in dem ein Front-End die Anfragen entgegennimmt und je nach Auslastung an unterschiedliche Back-End-Server zur Verarbeitung schickt. Die Auswirkungen eines erfolgreichen Request Smuggling Angriffs können verheerend sein, es können sensitive Informationen abgegriffen werden, ansonsten unerreichbare Schwachstellen ausgenutzt werden und die normale Benutzung der Webseite für alle Benutzer beeinflusst werden.

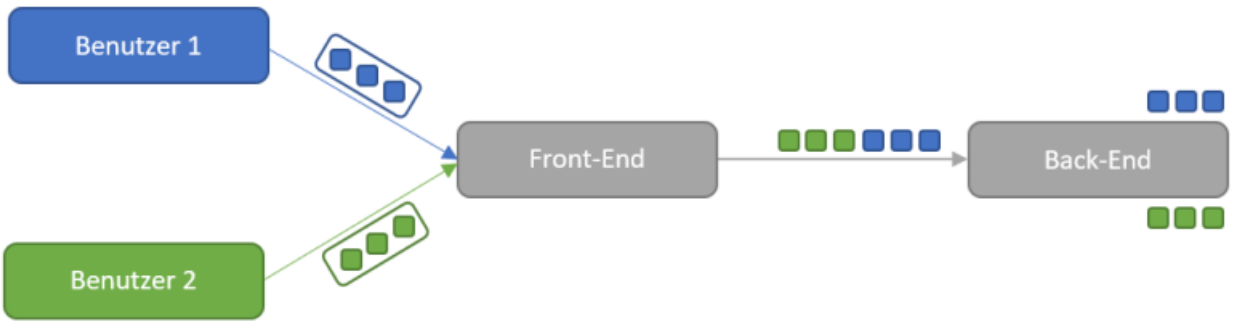
Im Normalfall wird bei HTTP/1.1 zwischen dem Front-End-Server und dem Back-End-Server die *Verbindung nicht nach jeder Anfrage beendet* und eine neue Verbindung aufgebaut. Dies wird gemacht, um Ressourcen zu sparen, da jeder neue Verbindungsaufbau Zeit braucht. Das heisst, dass das Einzige was in der Queue zwischen Front-End und Back-End die Anfragen voneinander aufteilt, sind die in einer Anfrage angegebenen Grössenangaben.

CONTENT-LENGTH UND TRANSFER-ENCODING HEADER

Es gibt *zwei unterschiedliche Arten*, wie in einer HTTP/1.1 Anfrage die Grösse des Bodys angegeben werden kann. Dies kann einerseits über den *Content-Length* Header und andererseits über den *Transfer-Encoding* Header geschehen. Der Content-Length Header gibt die Grösse des Bodys in Bytes an. Beim Transfer-Encoding Header mit dem Parameter *chunked* wird in sogenannten *Chunks* gearbeitet. Beispiel einer solchen Anfrage:

```
POST /test HTTP/1.1
Host: example.com
Transfer-Encoding: chunked
6
test=1
0
```

Dabei wird im Body als erstes die Grösse des Chunks in Bytes in Hexadezimal angegeben. Im Fall des Beispiels ist der Body *test=1* 6 Bytes gross. Ein Body kann grundsätzlich mehrere solche chunks mit ihrer jeweiligen Grössenangabe beinhalten. Ein chunked Request wird mit einem 0 und zwei Zeilenumbrüchen abgeschlossen.



HAUPTSCHWACHSTELLEN-TYPEN REQUEST SMUGGLING

Eine Request Smuggling-Attacke kann nun ausgeführt werden, indem es gelingt die Server in der Kette dazu zu bringen *unterschiedliche Grössenangaben* zu verarbeiten. Dabei werden den Servern mehrere Möglichkeiten gegeben Grössenangaben zu identifizieren, indem sowohl *Content-Length* wie auch *Transfer-Encoding* Header spezifiziert werden. Es gibt dabei drei unterschiedliche Arten wie Server anfällig sein können auf die Schwachstelle.

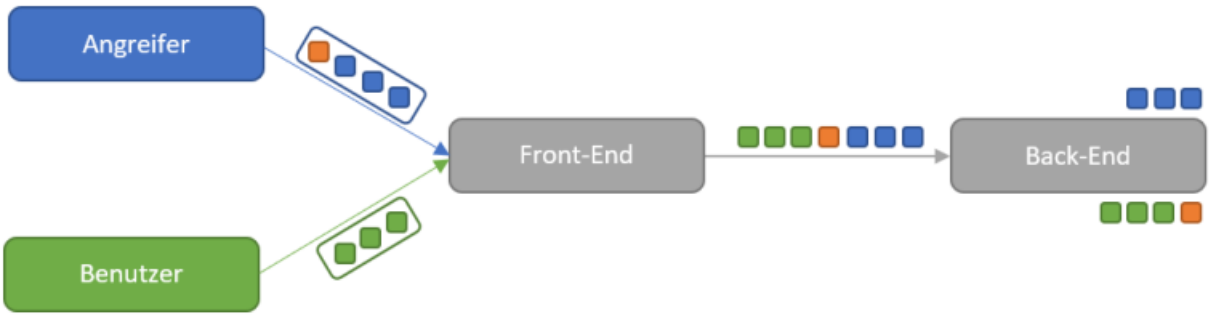
Bevor allerdings auf konkrete Angriffsmöglichkeiten eingegangen wird, hier ein Wort der Warnung: Das Testen dieser Schwachstellen kann *Auswirkungen auf Drittpersonen* haben, welche die getestete Webseite zur gleichen Zeit benutzen. Das Testen auf diese Schwachstelle sollte erst dann gegen produktive Umgebungen durchgeführt werden, wenn sich der Tester bereits ausführlich mit der Schwachstelle befasst hat und sich den Auswirkungen eines Request Smuggling bewusst ist. Um ein Gefühl für die Schwachstelle zu bekommen, werden die Labs der *Web Security Academy* von *Portswigger* empfohlen. Das Einlesen in dieses Thema ist angeraten, da Request Smuggling weitreichende Folgen haben kann. Im Beispiel von Slack wäre eine Übernahme des Session-Cookies von beliebigen Benutzern und somit das Lesen von beliebigen Nachrichten möglich. Die vollumfängliche Beschreibung dieser Schwachstelle ist in HackerOne Report 737140 zu finden.

Die nun folgenden Definitionen der Schwachstellentypen folgen der folgenden Notation:

{Grössenangabe Front}.{Grössenangabe Back}

Schwachstellentyp CL.TE

Im Fall von CL.TE ist es so, dass das Front-End die *Content-Length* der Anfrage für die Grössenangabe des Bodys verwendet und den *Transfer-Encoding* Header ignoriert. Das Back-End nimmt allerdings den *Transfer-Encoding* Header als Ausgangslage, um die Grösse des Bodys zu berechnen. In einem Beispiel würde dies wie folgt aussehen.



```
POST /test HTTP/1.1
Host: example.com
Content-Length: 13
Transfer-Encoding: chunked

0
--> Back-End liest bis hier aufgrund des 0 Chunks
Injected --> Front-End liest bis hier
```

In diesem Beispiel kennzeichnen die --> Kommentare und können für die Grössenberechnungen des Bodys ignoriert werden. Die Content-Length von 13 Bytes kommt zustande, da zusätzlich zu den im Beispiel sichtbaren Buchstaben noch die unsichtbaren Zeichen *\r* und *\n* für die beiden Zeilenumbrüche genutzt werden.

Da das Back-End die Verarbeitung der Anfrage früher beendet, als was das Front-End gesandt hat, bleibt der Rest der Anfragen in der Pipeline zwischen Front-End und Back-End. Sobald eine nächste Anfrage eintrifft, wird der noch in der Pipline vorhandene Rest der vorherigen Anfrage vor dieser neuen Anfrage hinzugefügt. Eine neue Anfrage würde im Back-End also wie folgt aussehen:

```
InjectedGET /test HTTP/1.1
Host: example.com
```

Schwachstellentyp TE.CL

Im Fall von TE.CL nutzt das Front-End den *Transfer-Encoding* Header und das Back-End verwendet den *Content-Length* Header. In einem Beispiel sieht das wie folgt aus.

```
POST /test HTTP/1.1
Host: example.com
Content-Length: 8
Transfer-Encoding: chunked

D
test=Injected --> Back-End liest bis und mit
test=
0
--> Front-End liest bis hier
```

Da das Back-End die Verarbeitung der Anfrage früher beendet, als was das Front-End gesandt hat, bleibt der Rest der Anfrage in der Pipeline zwischen Front-End und Back-End. Sobald eine nächste Anfrage eintrifft, wird der noch in der Pipline vorhandene Rest der vorherigen Anfrage vor dieser neuen Anfra-

ge hinzugefügt. Eine neue Anfrage würde im Back-End also wie folgt aussehen:

```
Injected
0


GET /test HTTP/1.1
Host: example.com
```

Schwachstellentyp TE.TE

Im Fall von TE.TE benutzen sowohl das Front-End wie auch das Back-End den *Transfer-Encoding* Header. Durch eine Manipulation des *Transfer-Encoding* Headers kann allerdings eines der Systeme dazu gebracht werden, den Header nicht mehr zu erkennen und auf die *Content-Length* zurückzufallen, während der andere Server dennoch noch den *Transfer-Encoding* Header benutzt. Je nachdem welches System denn *Transfer-Encoding* Header nicht mehr verwendet, kommt dann wiederum der Schwachstellentyp CL.TE oder TE.CL zum Einsatz.

Es gibt *unzählige Varianten*, wie der *Transfer-Encoding* Header verschleiert werden kann. Zum Beispiel wurde festgestellt, dass gewisse Server im Transfer-Encoding Header nur nach dem String *chu* suchen, um festzustellen, ob es ein chunked Request ist. Gepaart mit einem System, dass dies nicht so handhabt, führt der Header *Transfer-Encoding: chu* zu einer Diskrepanz zwischen den beiden Servern und öffnet die Möglichkeit für Request Smuggling-Angriffe. Weitere Beispiele den *Transfer-Encoding* Header zu verschleiern sind:

- Einfügen eines Leerzeichens zwischen dem : zwischen Transfer-Encoding und chunked
- Beginnen des Headers mit einem Leerzeichen
- Verwenden von zwei Transfer-Encoding Headern, wobei einer einen ungültigen Wert hat
- Einfügen eines Zeilenumbruchs vor dem : zwischen Transfer-Encoding und chunked

Ein Beispiel, wie mit einer Verschleierungstechnik des Transfer-Encoding Headers ein Request Smuggling-Angriff auf eine Webseite des U.S. Departments of Defense durchgeführt werden konnte, ist in HackerOne Report 526880 zu finden.

WEITERFÜHRENDE ANGRIFFE

Bei den drei oben gezeigten Schwachstellentypen handelt es sich um die Haupttypen von Request Smuggling-Angriffen auf denen weitere Attacken basieren. So wird es durch eine Request Smuggling-Schwachstelle zum Beispiel möglich *weitere Schwachstellen* auszunutzen, die ansonsten nicht ausnutzbar wären oder nur eine eingeschränkte Auswirkung haben. Beispiele dafür sind Angriffe, die in Headern stattfinden, die normalerweise nicht von einem Angreifer kontrolliert werden können oder XSS-Schwachstellen, die eigentlich nur einen Bereich betreffen, auf den ein Angreifer im Normalfall keinen Zugriff hat. Im HackerOne Report 955170 wird erläutert, wie eine normalerweise harmlose Verhaltensweise bei einem Redirect im Falle einer Request Smuggling-Schwachstelle zu einem Open Redirect führt und in diesem Fall dann zur Ausführung von vom Researcher kontrollierten JavaScript führte.

Wenn bei einem Request Smuggling-Angriff nicht wie bis anher gezeigt nur ein Teil einer Anfrage in die Pipeline hinzugefügt, sondern eine zweite vollständige, valide Anfrage hinzugefügt wird, ist es sogar möglich, die gesamte *Kommunikation zu desynchronisieren*. Denn dann denkt das Front-End es hat eine Anfrage weitergeleitet, das Back-End sieht allerdings zwei Anfragen und wird dementsprechend zwei Antworten schicken, damit erhält nun jeder Benutzer nicht die Antwort auf seine Anfragen, sondern die des vorgängigen Benutzers. Dies ist natürlich eine sehr *destruktive Vorgehensweise*, da damit *sämtliche Benutzer* dieser Front-End Back-End Verbindung davon betroffen sind.

Mit der Einführung des Protokolls HTTP/2 gibt es bei *Protokoll-Downgrades* zwischen dem Front-End und dem Back-End Server *weitere Angriffsflächen*, falls beim Downgrade der Anfrage nicht sauber umgewandelt wird. Darauf wird jedoch in einem eigenen Artikel zu HTTP/2 zu einem spätere Zeitpunkt genauer eingegangen.

HINWEISE FÜR TESTER

Aufgrund der weitreichenden Möglichkeiten dieser Schwachstelle kann das Testen bei einem Fehler zu Auswirkungen auf die gesamte Umgebung führen, das Testing sollte dementsprechend mit Sorgfalt angegangen werden. Mit einer aktuellen Version von

BurpSuite Pro wird bei der Durchführung eines *Active Scans* bereits auf Request Smuggling geprüft. Dabei wird ein Vorgehen gewählt, bei dem die *Auswirkung* auf andere Benutzer *möglichst klein gehalten* wird. Am einfachsten machbar ist dies, in dem mit *bewusst ausgelösten Time-Outs* gearbeitet wird. Zum Beispiel, wenn eine CL.TE Schwachstelle vorhanden ist, kann mit folgender Anfrage ein Time-Out ausgelöst werden:

```
POST /test HTTP/1.1
Host: example.com
Content-Length: 9
Transfer-Encoding: chunked

6
x=test --> Front-End liest bis hier
X --> Back-End erwartet einen weiteren Chunk oder
den Abschluss der Chunks
```

Da das Back-End nur den ersten Chunk erhält aber nicht entweder das abschliessende 0 oder einen weiteren Chunk wartet das Back-End auf mehr Inhalt, was zu einem Time-Out führt.

Für eine TE.CL Schwachstelle würde das Auslösen eines Time-Outs aufgrund einer Request Smuggling-Schwachstelle wie folgt aussehen:

```
POST /test HTTP/1.1
Host: example.com
Content-Length: 11
Transfer-Encoding: chunked

0 --> Front-End liest bis hier

x=test --> Back-End erwartet aufgrund des Content
-Length Headers Inhalt bis hier
```

Da das Back-End nur den Inhalt bis zum 0 erhält aber eine grössere *Content-Length* vorgegeben hat, wartet das Back-End auf mehr Inhalt, was zu einem Time-Out führt.

Es ist wichtig die Schwachstellen immer in der Reihenfolge CL.TE und dann TE.CL zu prüfen, da es bei einer vorhandenen CL.TE Schwachstelle mit einem TE.CL Time-Out Anfrage zu einer Störung von anderen Benutzern der Webapplikation kommen kann.

Wenn durch den *ActiveScan* von Burp oder durch manuelles Testen eine Request Smuggling-Schwachstelle identifiziert wurde, bietet BurpSuite Pro den *Turbo Intruder* an, mit dem die identifizierten Schwachstellen geskriptet und ausgenutzt werden können. Ein Beispiel dafür ist in HackerOne Report 498052 zu einer CL.TE Schwachstelle in New Relic beziehungsweise F5 zu finden.

Ebenfalls zu beachten ist, dass die *Möglichkeit von False-Negatives* besteht, wenn man die Schwachstelle auszunutzen versucht, da es sein kann, dass die erhoffte Antwort nicht beim Tester landet, sondern bei einem nichtsahnenden Benutzer der Webseite. Weiter ist zu beachten, dass es auch zu False-Negatives kommen kann, wenn in einer Entwicklungs- oder Testumgebung getestet wird, da diese eventuell nicht hinter dem gleichen Load-Balancing-Setup liegen wie die produktive Umgebung.

GEGENMASSNAHMEN

Es sollte geprüft werden, ob im eingesetzten Produkt diese Schwachstelle bereits bekannt ist und ein Update/Patch dafür zur Verfügung steht. Da Request Smuggling Schwachstellen bei einer unterschiedlichen Interpretation von Request-Grössen zwischen einem Front-End und einem Back-End entstehen, sollte da angesetzt werden. Das *Front-End sollte Anfragen normalisieren*, die sowohl die Header *Content-Length* und *Transfer-Encoding* chunked enthalten, so dass nur noch einer dieser Header verwendet wird. Wenn das *Back-End* dennoch jemals eine Anfrage erhält, die sowohl *Content-Length* wie auch *Transfer-Encoding* chunked Header enthält, sollte die *Anfrage verworfen* und die *TCP-Verbindung geschlossen* werden. Falls die Schwachstelle aufgrund eines HTTP/2-Protokoll-Downgrades zustande kam, sollte wo möglich durchgehend nur HTTP/2 verwendet werden, damit es gar nicht erst zu dieser Schwachstelle kommen kann.

FAZIT

Bei *Request Smuggling* handelt es sich um eine sehr mächtige Schwachstelle mit weitreichenden Auswirkungen. Es wird damit möglich sensitive bzw. vertrauliche Informationen abzufangen und Schwachstellen auszunutzen, die ansonsten nicht ausnutzbar wären. Es ist dementsprechend wichtig beim Testen diese Auswirkungen im Hinterkopf zu behalten, denn wenn etwas falsch gemacht wird, können potentiell alle Benutzer der geprüften Webseite betroffen sein. Es sollte zudem beachtet werden, dass ein Test auf Request Smuggling nicht immer eine hundertprozentige Aussage darüber machen kann, ob diese Schwachstelle tatsächlich vorhanden ist oder nicht.

Andrea Hauser

WIR SIND IHR PARTNER FÜR PROFESSIONELLE

CYBERSECURITY SERVICES

EINFACH ONLINE BESTELLEN

MARIUS ELMIGER

HONEYTOKENS ALS MINESWEEPER FÜR ANGREIFER

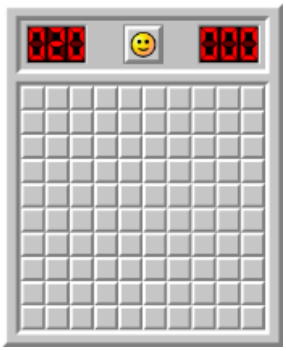
Das *Defender-Dilemma* ist wohl allen bekannt, es besagt, dass Angreifer nur eine Sicherheitslücke in einem IT-Unternehmen ausnutzen müssen. Wobei ein Verteidiger alle Lücken schliessen muss, was ein Ding der unmöglich ist. Diese negative Perspektive ist nicht förderlich, um ein kreatives Defenders-Mindset aufbauen zu können. In diesem Beitrag wollen wir daher den Fokus auf das Dilemma der Angreifer richten und wie dieses am Beispiel von *Honeytokens* ausgenutzt werden kann.

Ein Angreifer betritt das Spielfeld eines Defender, sobald eine Kompromittierung, meist eines Endpunktes, durch eine Angriffsvariante stattgefunden hatte. Die Angreifer haben nun verschiedene Möglichkeiten das Spielfeld zu erkunden, vorstellbar wie beim Game Minesweeper welches wahrscheinlich nur noch die ältere Generation unter euch kennt.

Die Aufgabe als Defender besteht darin die Angreifer so schnell wie möglich zu erkennen und zu blockieren, bevor diese auf kritische Daten zugreifen können. Um das zu erreichen, setzen wir heutzutage meistens eine Vielzahl an Sicherheitstools ein, welche unter anderem Logs von unseren Endpunkten sammeln. Anhand von unseren durchgeführten Red Teams können wir jedoch sagen, dass zwar unsere Aktionen meistens aufgezeichnet wurden, jedoch im Datenlärm untergangen sind. Woran kann, dass liegen? Ein Grund könnte sein, dass wir als Defender dazu tendieren zu wenig kreativ zu sein. Wir arbeiten Listen ab, um unsere Systeme sicherer zu machen aber achten zu wenig auf das Spielfeld auf welchem sich ein Angreifer bewegt. Passend dazu ist auch John Lambert's (2015) Zitat:

> Defenders think in lists. Attackers think in graphs. As long as this is true, attackers win.

Wie können wir nun das Spielfeld für Angreifer schwieriger gestalten? Angreifer versuchen in einem ersten Schritt meistens weitere Informationen über unsere IT-Umgebung zu sammeln. Diese Aktion findet entweder auf dem Endpunkt oder über das Netzwerk zu Diensten wie zum Beispiel *Active Directory*, *File Server* oder andere *Umsysteme* statt. Da wollen wir nun mit einer möglichen Methode genannt *Honeytokens* ansetzen, um aufzuzeigen, dass wir mit dieser effektiven Art in der Lage sein können, frühzeitig auf ein erstes Indiz reagieren zu können. Dafür legen wir auf unserem Spielfeld gezielt Fallen aus, vorstellbar wie bei Minesweeper die Minen.



Es ist wichtig bei der Platzierung der *Honeytokens* sich in die *Denkweise der Angreifer* zu versetzen. Umso kreativer wir diese Fallen stellen umso eher werden diese durch die Angreifer ausgelöst. Bei einer Aktivierung haben wir nun den Vorteil durch unsere zahlreichen Tools festzustellen von welcher Komponente (z.B. User, IP-Adresse, Hostname) ein Angriff ausgegangen ist. Nun liegt es an uns so schnell wie möglich zu reagieren, um die Angreifer von unserem Spielfeld zu vertreiben.

Honeytokens gibt es in *verschiedenen Variationen*. Im nachfolgenden Kapitel beschreiben wir ein Beispiel anhand von einem *Active Directory Honeytoken*.

HONEYTOKEN IDEE FÜR ACTIVE DIRECTORY

Active Directory ist für Angreifer ein idealer Ort, um Informationen über die IT-Umgebung zu sammeln. Vergleichbar mit Google Maps, wir suchen nach einem guten Restaurant oder Sehenswürdigkeiten, Angreifer suchen im Active Directory nach einem einfachen Pfad, um an *privilegierte Objekte* heranzukommen. Um Active Directory auszulesen, existieren viele Tools wie zum Beispiel dsquery.exe, dsa.msc, SharpHound, ADExplorer, PowerView und viele mehr auf welche wir in diesem Artikel jedoch nicht speziell eingehen werden. Wir sind nämlich eher interessiert an der Abfrage selbst welche über LDAP stattfindet und welche wir mit Hilfe von *System Access Control Lists* (SACL) auf einem Domain Controller aufzeichnen können. SACL werden im Vergleich zu *Discretionary Access Control List* (DACL) zu wenig verwendet. Anders als eine DACL welche über SIDs den Zugriff auf ein Objekt kontrolliert, ermöglicht die SACL den Zugriff auf die *Audit ACE* eines Objektes. Eine Audit ACE beschreibt, ob eine gewisse Aktion auf ein Objekt erfolgreich war oder nicht. Zu empfehlen für eine tiefere Recherche zum Thema ACL ist das An ACE Up the Sleeve Dokument, welches von der Firma SpecterOps veröffentlicht wurde.

Für unseren Active Directory Honeytoken werden wir von den zwei SACL Kategorien *Object Read* und *Object Write* nur das Object Read verwenden. Wir können damit aufzeichnen, ob Active Directory ausgelesen wird. Die Vorgehensweise sieht wie folgt aus:

1. In einem ersten Schritt erstellen wir einen neuen Benutzer mit einem komplexen Kennwort in Active Directory. Die Namenskonvention sollte sich an die bereits vorhandene Objekte richten, um zu vermeiden, dass ein Angreifer das Konto schnell als Honeytoken identifizieren kann. Des Weiteren gilt bereits zu überlegen nach was für einem Namen oder Prefix ein Angreifer in einem Active Directory suchen würde wie zum Beispiel nach adm*, admin* oder svc*

2. Nun sollten wir uns in den nächsten Tagen mehrmals mit dem erstellten Honeytoken anmelden um die LastLogon-Attribute in Active Directory zu befüllen. Angreifer versuchen nämlich Honeytokens zu erkennen indem sie Konten herausfiltern, die sich nie angemeldet haben oder deren Kennwort am selben Tag wie die LastLogon-Attribute festgelegt wurden.

3. Nun fügen wir eine neuen SACL über das Security Tab -> Advanced -> Auditing hinzu:

- Principal: Everyone
- Applies to: The object only
- Permissions: Read all properties

4. Damit die Domain Controller einen Event durch unsere SACLs Einstellung erzeugen können, müssen wir die Advanced Audit Policy wie folgt konfigurieren:

- DS Access – Audit Directory Service Access: Success (Mindestens Success)
- Vorteilhaft wird diese Einstellung über GPOs vorgenommen und muss auf alle Domain Controller appliziert werden.

5. Nun können wir unseren Honeytoken testen in dem wir eine LDAP-Abfrage auf den erstellten Honeytoken ausführen. In unserem Beispiel haben wir dsquery und SharpHound verwendet.

- Abfrage mit dsquery nach allen AD Benutzern welche mit adm beginnen
- Diese Abfrage erzeugt zwei Security Eventlog Einträge auf dem Domain Controller mit der Event ID: 4662.
- Die Abfrage mit SharpHound erzeugt mit unserer Read All Properties SACL-Einstellung vier Einträge, da SharpHound mehrere Properties abfragt.

6. Als letzter Schritt muss nun der Event 4462 von der verwendeten Sicherheitslösung eingesammelt und ein Alarm ausgelöst werden.

Von nun an können wir feststellen, ob Angreifer das gesamte Active Directory auslesen oder in unserem Beispiel spezifisch nach adm Accounts suchen. Zu beachten gilt das *False-Positives* auftreten können, falls der erstellte Honeytoken in einer OU angelegt wurde in welchen legitime Administratoren oft Abfragen tätigen oder durch 3rd Party Applikationen wie zum Beispiel ein Identity Management System. *Ausnahmen* können daher erforderlich sein, um False -Positives zu vermeiden.

WEITERE HONEYTOKEN-IDEEN FÜR SACLS

In Windows gibt es eine Vielzahl an *Securable Objects* wie zum Beispiel:

- Network Shares
- Registry Keys
- Dateien und Verzeichnisse auf dem NTFS-Dateisystem
- Printers
- Processes
- Active Directory Objects

Auf alle diese Objekte können SACLs gesetzt werden. Dateien auf Shares oder auch auf Clients sind hervor-

ragend geeignet für Honeytokens und erzeugen wie am Beispiel Active Directory auch ein Event mit der ID 4462. Vorausgesetzt die *Advanced Audit Policy* ist korrekt konfiguriert. Der Kreativität ist schlussendlich kein Ende gesetzt. Als Start für die Dateinamen kann man sich an der *PowerView* Funktion *Find-InterestingDomainShareFile* orientieren welche von Angreifern gerne verwendet wird. Die Funktion sucht standardmässig nach Namen wie pass, sensitive, secret, admin, login oder unattend*.xml.

Weitere Honeytoken-Ideen

Zahlreiche Webseiten beschreiben verschiedene Arten von Honeytoken-Implementierung. Nachfolgend präsentieren wir einige Links:

- *Canary Tokens* oder *Canary Tools* bieten eine automatisierte Möglichkeit, Honeytokens zur Simulation von Netzwerkdiensten einzusetzen. Wenn Angreifer das Netzwerk nach interessanten Diensten wie File-Servern oder Webdiensten durchsuchen, wird ein Alarm ausgelöst

- Microsoft beschreibt, wie man ein *Azure Key Vault Honeytoken* erstellt, indem ein Alarm ausgelöst wird, wenn ein Angreifer versucht, Schlüsselmaterial auszulesen

- Rene Feingruber beschreibt in seinem Blogbeitrag verschiedene Arten von Honeytokens. Besonders kreativ sind die Honeytoken-Ideen zu den Tools BloodHound, PowerView und Responder, die von Angreifern häufig verwendet werden

- Nikhil Mittal beschreibt in seinem Blogbeitrag, wie Angreifer Honeytokens identifizieren können. Wie bereits erwähnt sollten wir uns in die Lage eines Angreifers versetzen um realistische Fallen auslegen zu können. Was würde daher ein Attacker als Täuschung ansehen und ignorieren?

- Sean Metcalf beschreibt detailliert, wie man ein normal aussehender Honeytoken in Active Directory erstellt

FAZIT

Das Erstellen effektiver *Honeytokens* in IT-Umgebungen erfordert sowohl die *Denkweise eines Angreifers* als auch *Kreativität*. Wie wir gelernt haben, können Honeytokens für nahezu jedes Szenario entwickelt werden und kann helfen einen Angreifer in der Frühphase eines Angriffs zu enttarnen. Dadurch kann der Incident möglicherweise schneller bearbeiten und gestoppt werden. Es sollten mehrere Honytokens in einer IT-Umgebung erstellt werden, welche anhand der Angriffsstadien ausgelegt werden sollten. Zu beachten gilt, dass Honeytokens die IT-Umgebung nicht gefährden sollten. Daher empfehlen wir keine privilegierten Zugriffsrechte an Honeytokens zuzuweisen. Wie bei allem in der IT sollten die Honeytokens getestet werden, ob sie wie erwartet funktionieren. Und schliesslich sollten man sich nicht nur auf Honeytokens verlassen. Sie sind jedoch eine wirksame Technik, die Defender einsetzen können, um das *Dilemma des Angreifers* zu vergrössern.

Marius Elmiger

ROBUSTHEIT BEDEUTET AUCH SICHERHEIT

Bildquelle: https://pixabay.com/images/id-6768469/

SCIP MONTHLY SECURITY SUMMARY

IMPRESSUM

ÜBER DEN SMSS

Das *scip Monthly Security Summary* erscheint monatlich und ist kostenlos.

Anmeldung: smss-subscribe@scip.ch
Abmeldung: smss-unsubscribe@scip.ch

Informationen zum Datenschutz.

Verantwortlich für diese Ausgabe:
Marc Ruef

Eine Haftung für die Richtigkeit der Veröffentlichungen kann trotz sorgfältiger Prüfung durch die Redaktion des Herausgebers, den Redaktoren und Autoren nicht übernommen werden. Die geltenden gesetzlichen und postalischen Bestimmungen bei Erwerb, Errichtung und Inbetriebnahme von elektronischen Geräten sowie Sende- und Empfangseinrichtungen sind zu beachten.

ÜBER SCIP AG

Wir überzeugen durch unsere Leistungen. Die scip AG wurde im Jahr 2002 gegründet. Innovation, Nachhaltigkeit, Transparenz und Freude am Themengebiet sind unsere treibenden Faktoren. Dank der vollständigen Eigenfinanzierung sehen wir uns in der sehr komfortablen Lage, vollumfänglich herstellerunabhängig und neutral agieren zu können und setzen dies auch gewissenhaft um. Durch die Fokussierung auf den Bereich Information Security und die stetige Weiterbildung vermögen unsere Mitarbeiter mit hochspezialisiertem Expertenwissen aufzuwarten.

Weder Unternehmen noch Redaktion erwähnen Namen von Personen und Firmen sowie Marken von fremden Produkten zu Werbezwecken. Werbung wird explizit als solche gekennzeichnet.

scip AG
Badenerstrasse 623
8048 Zürich
Schweiz

+41 44 404 13 13
www.scip.ch