Eintrittspforte für Cyberkriminelle: Software weist meist Sicherheitslücken auf. Das automatisierte Prüfverfahren eines Max-Planck-Teams klopft Programme effektiv und effizient darauf ab.

ILLUSTRATION: ALESSANDRO GOTTARDO

# SCHLUPFLOCH IM PROGRAMM

*TEXT:*
*THOMAS BRANDSTETTER*



Angriffe auf Software verursachen nicht nur Schäden in Milliardenhöhe, sondern bedrohen auch die Privatsphäre von Nutzerinnen und Nutzern. Cyberkriminelle dringen dabei stets durch Sicherheitslücken in Programme ein. Marcel Böhme und sein Team am Max-Planck-Institut für Sicherheit und Privatsphäre sind angetreten, die Einfallstore für die Angreifer zu blockieren – und haben auch Unternehmen wie Google von ihrem Ansatz überzeugt.

Programmieren ist ein kreativer Prozess. Er beginnt mit der Idee des Programmierers, eine gewünschte Funktionalität umzusetzen, und endet mit einem funktionstüchtigen Code. Doch was funktioniert, ist noch lange nicht sicher. Dabei steckt die Tücke oft im Detail, und sie kann sich als große Gefahr entpuppen. So basierte der Heartbleed-Fehler, durch den 2014 unter anderem Zugangsdaten zu zahlreichen Onlinediensten öffentlich wurden, auf einer Sicherheitslücke in einer Software mit einer sehr überschaubaren Aufgabe: Das kleine Programm hieß Heartbeat und löste das Problem, dass ein Browser beim Surfen im Internet oder beim Onlinebanking über eine gesicherte Verbindung manchmal noch verschlüsselte Daten sendete, obwohl die geschützte Verbindung längst abgebrochen war. Über Heartbeat kann der Browser beim Server nachfragen, ob die gesicherte Verbindung noch besteht. Zu diesem Zweck sendet die Software regelmäßig eine Zeichenkombination samt Angabe der Zeichenzahl an den Server und erwartet die gleiche Zeichenkombination als Antwort. Der Entwickler nahm natürlich an, dass der Browser immer die korrekte Zeichenzahl angibt. Anders als bei anderer Software üblich baute er allerdings keinen Mechanismus ein, um das zu überprüfen. Und diese Lücke nutzte der Angreifer aus; er manipulierte Heartbeat so, dass das Programm eine kurze Zeichenkette mit der maximalen Längenangabe verschickte. Als der Server die Zeichenzahl dann aus dem Speicher auslas, kopierte er wesentlich mehr Daten, darunter auch sensible Informationen, als die ursprüngliche Zeichenkombination umfasste – Fachleute sprechen in diesem Fall von Speicherkorruption.

So steht der Absicht des Programmierers, ein bestimmtes Problem zu lösen, allzu oft die Perspektive eines böswilligen Hackers gegenüber. Und der fragt sich, wie er den Code der Software für seine eigenen Zwecke nutzen kann. Berüchtigt sind auch die Angriffe von Cyberkriminellen, die sämtliche Dateien verschlüsseln, um für die Freigabe ein Lösegeld zu verlangen. Auf solche Ransomware-Attacken ist der größte Teil des wirtschaftlichen Schadens durch Cyberkriminalität zurückzuführen. Dieser belief sich im Jahr 2021 dem Branchenverband Bitkom zufolge allein in Deutschland auf mehr als 220 Milliarden Euro.

Verschärft wird die Problematik der IT-Sicherheit noch dadurch, dass moderne Softwaresysteme nur selten von einer Person oder zumindest von einer einzigen Firma entwickelt werden. Stattdessen sind sie oft aus einer Vielzahl von Komponenten zusammengebaut, die aus unterschiedlichen Quellen stammen. Und jede dieser Komponenten besteht ihrerseits wieder aus einzelnen kleinen Beiträgen unabhängiger Programmierer, die es mit der Sicherheit unterschiedlich genau nehmen.

„Es gibt sehr viele kleine Open-Source-Softwaresysteme, die von Menschen in ihrer Freizeit vielleicht einmal an einem Nachmittag entwickelt wurden, in den letzten zwanzig Jahren aber unheimlich kritisch und fundamental für unsere digitale Ökonomie geworden sind“, sagt Marcel Böhme, der am Max-Planck-Institut für Sicherheit und Privatsphäre die Gruppe für Software Security leitet. Außer den ursprünglichen Entwicklern selbst fühle sich jetzt aber niemand mehr verantwortlich für die Sicherheit all dieser Einzelkomponenten. Und auch die Entwickler selbst hätten nach so langer Zeit teilweise einfach keine Lust mehr, ihre Programme weiterzuentwickeln. Trotzdem bieten öffentlich verfügbare Softwareelemente gerade in puncto Sicherheit einige Vorteile, da viele Fachleute sie überprüfen können. Auch aus diesem Grund verwenden Unternehmen wie etwa Google Open-Source-Code. Um in der eigenen Software Sicherheitslücken aufzuspüren, arbeitet Google jetzt mit Böhme und seinem Team zusammen.

Die Crux der Softwaresicherheit ist, dass die Kette der Programmteile nur so gut geschützt ist wie ihr schwächstes Glied. Und das macht auch große, kommerzielle Systeme anfällig für Angriffe. Um das ganze System zu übernehmen, kann es für einen findigen Hacker genügen, eine einzige schlecht geschützte Komponente aufzuspüren. „Das liegt oftmals daran, dass dieser Aspekt zum Zeitpunkt der Entstehung des Programms noch nicht so wichtig war“, sagt Böhme. „Viele große Sicherheitslücken, die wir heutzutage in weltweit verbreiteten Systemen finden, lassen sich auf solche ‚kleinen‘ Sicherheitslücken zurückführen.“ Die Speicherkorruption sei dabei besonders kritisch. Sie lässt sich nicht nur ausnutzen, um – wie im Fall von Heartbleed – Daten auszuspionieren und zu klauen, sondern auch, um Befehle in ein Programm einzuschmuggeln, mit denen ein Angreifer im schlimmsten Fall die Kontrolle über den Rechner übernehmen kann. Denn so wie sich mehr aus einem Speicher auslesen lässt als vorgegeben, lässt sich auch mehr in ihn hineinschreiben, als von einem Programm vorgesehen ist – wenn die Software nicht so programmiert ist, dass sie die Vorgaben für die angeforderte oder übertragene Datenmenge überprüft. Um derartige Probleme im Programmcode aufzudecken, wer-

## AUF DEN PUNKT GEBRACHT

Softwareentwickler müssen darauf achten, dass Programme nicht angreifbar sind. Doch Software, die nicht auf Sicherheitslücken überprüft wird, weist häufig kritische Fehler auf.

Forschende des Max-Planck-Instituts für Sicherheit und Privatsphäre haben die automatisierte Suche nach Sicherheitslücken durch das Greybox Fuzzing erheblich beschleunigt.

Unternehmen wie Google, Bosch oder Oracle Labs nutzen die Methode bereits und entdecken dabei immer wieder neue Fehler.

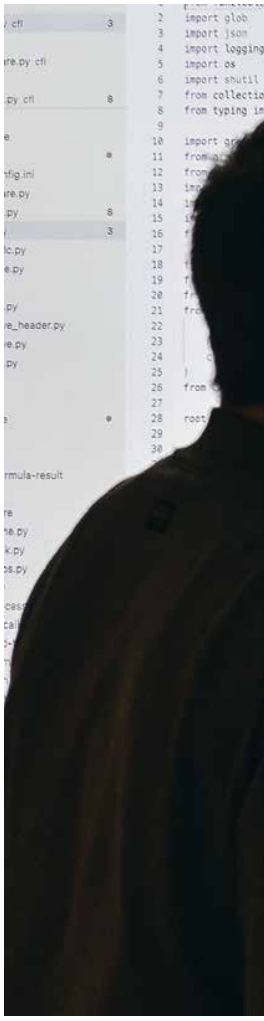


FOTO: FRANK VINKEN FÜR MPG

Viele Augen: Je mehr Menschen Programme prüfen, desto eher werden sicherheitsrelevante Fehler entdeckt – vor allem wenn es Fachleute sind wie in Marcel Böhmes Team, die prüfen. Automatisierte Tests beschleunigen die Suche jedoch.



## „Das Greybox Fuzzing vereint das Beste aus zwei Welten."

*MARCEL BÖHME*

den aktuelle Systeme, auch während sie sich bereits im Einsatz befinden, über Wochen, Monate und teilweise auch Jahre hinweg aufwendig auf kritische Lücken getestet – und gegebenenfalls durch den Softwarenotdienst der Sicherheitsupdates repariert. Das gilt auch für Software, die viele Menschen zu Hause nutzen. Würde sich bei Google Chrome etwa eine Sicherheitslücke auftun, über die Malware verbreitet werden kann, wären weltweit unzählige Nutzerinnen und Nutzer betroffen, die etwa zu Opfern von Ransomware-Attacken werden könnten. „Es könnten aber auch beispielsweise persönliche Daten, Passwörter oder das Browserverhalten gestohlen werden", warnt Böhme. „Die Wahrscheinlichkeit, dass das bei Google Chrome passiert, ist glücklicherweise sehr klein", meint er. „Schließlich hat Google viele Möglichkeiten, seine Systeme zu schützen."

Um die gefährlichen Mängel in einem Programm zu finden, gibt es eine Reihe von Ansätzen. Im einfachsten Fall sehen sich Menschen die Software genau an und suchen nach Fehlern. „Aber Maschinen, die das automatisch machen, erhöhen die Erfolgschancen", sagt Böhme. Bei den automatischen Verfahren wiederum wird zwischen statischer Analyse und dem sogenannten Fuzzing unterschieden. Statische Analyseverfahren untersuchen zunächst den Code eines Programms und erstellen daraus ein Modell, das sein Verhalten beschreibt. Marcel Böhme erklärt das in einem Vergleich mit der Biologie. In einem Zweig dieser Disziplin bauen Bioinformatiker etwa eine Zelle am Computer nach und simulieren so das Zusammenspiel von deren verschiedenen Komponenten nach den biologischen Regeln. „Ähnlich kann

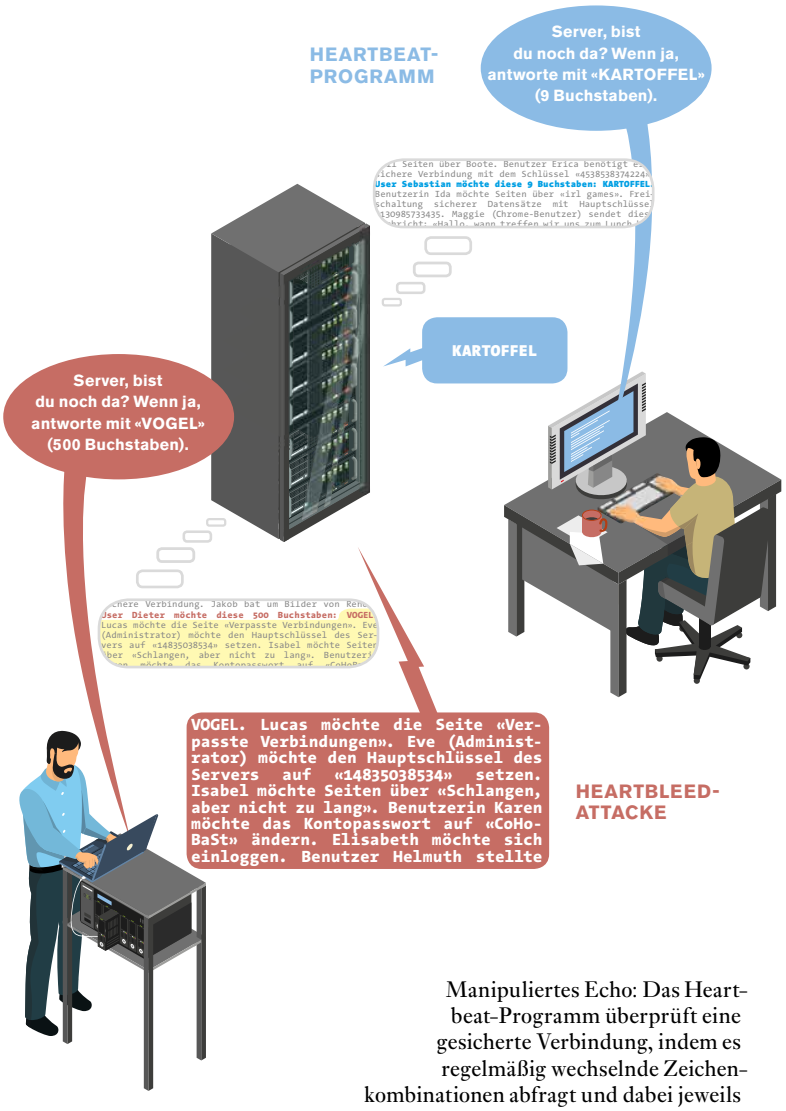
→

man ein Modell eines Programms erstellen“, erklärt Böhme. „Dabei simuliert man sein Verhalten aufgrund der syntaktischen und semantischen Regeln, nach denen Software geschrieben wird.“ Auf Basis dieses Modells wird dann versucht, alle erdenklichen Eingaben vorauszusehen und letztlich zu beweisen, dass keine davon zu einem kritischen Fehler führen kann. Wie bei der Zelle im Computer fehlt jedoch auch beim Modell eines Programms die reale Umwelt – bei einem Programm sind das alle anderen Programme, mit denen das getestete Programm kommuniziert.

## Systematische Zufallseingaben

Beim Fuzzing, dem sich Böhme und sein Team verschrieben haben, wird das Programm dagegen unter realen Bedingungen ausgeführt und mit möglichst vielen, zufällig generierten Eingaben konfrontiert. „Fuzzer simulieren quasi einen Nutzer, der nicht das macht, was sich der Programmierer vorgestellt hat“, erläutert Böhme. In der reinsten Form, dem Blackbox Fuzzing, kommt den Zusammenhängen im Code des untersuchten Programms keinerlei Bedeutung zu, und Fehler werden ausschließlich durch zufällige Eingaben entdeckt. Allerdings testet der Blackbox Fuzzer Programmteile, die auch bei der Nutzung häufig ausgeführt werden, wesentlich öfter als nötig, was nicht sehr effizient ist. Aber schlimmer noch: Selten gefragte Programmteile testet auch die Blackbox kaum oder im schlimmsten Fall gar nicht, sodass Sicherheitslücken darin unentdeckt bleiben. Das Problem lässt sich mit dem Whitebox Fuzzing lösen: Dabei wird zusätzlich auch der Code analysiert und ähnlich wie bei den statischen Methoden in ein formales Modell umgewandelt, das dann systematisch bis in den letzten Winkel abgeklopft wird. „Das ist sehr effektiv, dauert aber für moderne Programme viel zu lange“, sagt Böhme.

Zumindest am Anfang der Fehlersuche ist das zufällige Generieren von Eingabewerten also effizienter als die systematische Exploration des Programmverhaltens. Im Laufe des Testens holt der Whitebox-Ansatz allerdings auf, weil er dazulernt und sich bereits getestetes Verhalten nicht noch einmal ansehen muss. Dem Blackbox-Ansatz hingegen ist es egal, ob er ein bestimmtes Verhalten bereits gesehen hat oder nicht. „Für einen Wissenschaftler ist es schon irgendwie kontraintuitiv, wenn ein Beschuss mit rein zufällig generierten Eingaben bessere Resultate liefert als eine tiefgehende Analyse“, sagt Böhme. „Am Ende meiner Doktorarbeit habe ich deshalb begonnen, mich für die Frage zu interessieren, ob sich dieses Verhalten erklären lässt.“ Ein Faktor neben der Sicherheit ist dabei der Zeitaufwand. Während ein Whitebox-Ansatz gerade mal eine oder zwei Eingaben pro Sekunde schafft, sind es mit einem Blackbox Fuzzer ohne Weiteres hunderttausend zufällig generierte Eingaben pro Sekunde. „Wir haben herausgefunden, welcher Ansatz unter welchen Bedingungen am besten funktioniert“, erzählt




Manipuliertes Echo: Das Heartbeat-Programm überprüft eine gesicherte Verbindung, indem es regelmäßig wechselnde Zeichenkombinationen abfragt und dabei jeweils die Zeichenzahl angibt. Cyberkriminelle haben die Software 2014 dazu gebracht, deutlich mehr Zeichen aus dem Speicher eines Servers auszulesen, als die erbetene Antwort umfasst. So haben sie sensible Daten abgegriffen.

GRAFIK: GCO NACH HTTPS://XKCD.COM/1354/, ISTOCK

Böhme. „Das hat uns dann zum Greybox Fuzzing geführt, das gewissermaßen das Beste aus zwei Welten vereint.“ Ein Greybox Fuzzer erzeugt Eingaben genauso schnell wie ein Blackbox Fuzzer, nutzt aber wie ein Whitebox Fuzzer zusätzliches Feedback über die bereits ausgeführten Teile des Programms. Damit vermeidet ein Greybox Fuzzer wiederholte Tests derselben Softwareelemente, die den ganzen Prozess verlangsamen. Es entgehen ihm aber auch keine Softwareteile mit Nischenfunktionen.

Der Greybox Fuzzer ist dabei ein regelrechter Fehlerfuchs: „Wenn eine Software dem Fuzzing zum ersten Mal unterzogen wird, finden wir durchschnittlich zwei bis drei Fehler inklusive Sicherheitslücken pro Tag“, sagt Böhme. „Nach ein paar Wochen reduziert sich das auf drei bis vier neue Fehler pro Woche und

bleibt dann konstant, weil ja immer wieder neue Fehler eingeführt werden." Die Kombination von Effizienz und Sicherheit überzeugt auch Techunternehmen: Allein bei Google sind heute hunderttausend Rechner dafür reserviert, Greybox Fuzzer auszuführen und damit rund um die Uhr über fünfhundert Softwareprojekte zu testen.

Bei den am Max-Planck-Institut entwickelten Fuzzern handelt es sich ausschließlich um Open-Source-Anwendungen, die also frei zugänglich im Netz zu finden sind. „Wir stellen sie damit auch kleinen Programmierern zur Verfügung, um ihnen die Fehlersuche in ihren eigenen Programmen zu erleichtern", sagt Böhme. Darüber hinaus werden aber auch größere Open-Source-Projekte gescannt. Erst kürzlich haben Böhmes Fuzzer etwa eine schwerwiegende Sicherheitslücke in OpenSSL aufgedeckt, einer freien Software zur verschlüsselten Kommunikation in Browsern und E-Mail-Anwendungen. „Die Sicherheitslücke, die unser Team gerade gefunden hat, hätte es einem Angreifer erlaubt, Rechner zu übernehmen, von denen verschlüsselte E-Mails verschickt werden", erklärt der Informatiker.

## Sicherheit für das Internet der Dinge

Trotz zahlreicher Kooperationen mit großen Unternehmen wie Bosch oder Oracle Labs bleibt Google der wichtigste Kooperationspartner für Marcel Böhmes Forschungsgruppe. Der amerikanische Techgigant hat großes Interesse an der Sicherheit von Open-Source-Projekten, da sie auch essenzielle Bestandteile seiner eigenen Produkte darstellen. „Für uns wiederum ist die Zusammenarbeit mit Google interessant, weil sie eben Marktführer auf diesem Gebiet sind und über gewaltige Ressourcen verfügen, zu denen wir sonst keinen Zugang hätten", erklärt Böhme. „Das ist eine Art Symbiose." Denn auch wenn Google die frei zugänglichen Fuzzer ohnehin nutzen könnte, ist der Konzern durch die enge Kooperation immer nahe am aktuellen Stand der Entwicklung. Bei allen aktuellen Erfolgen will sich Böhme in Zukunft aber auch neuen Herausforderungen stellen. Schließlich krempeln Digitalisierung und künstliche Intelligenz die Welt der Datenverarbeitung gerade gehörig um und verlangen dabei auch nach völlig neuen Sicherheitskonzepten. So gibt es etwa unter dem Schlagwort Industrie 4.0 einen Trend weg von großen, zentralen Recheneinheiten und hin zu vielen kleinen Geräten, die über verschiedene Sensoren verfügen und damit kleinere Aufgaben erfüllen. „Die haben in der Regel eher wenig Rechenleistung, weshalb die Sicherheit während der Entwicklung solcher Systeme oft sehr geringe Relevanz hat", erklärt Böhme. Solche kleinen Einheiten tauschen häufig Daten mit anderen Geräten aus, etwa um von diesen die eigentlichen Berechnungen durchführen zu lassen. „Um im großen Maßstab sicherzustellen, dass die kleinen Geräte in diesem Internet der Dinge richtig funktionieren, könnte man etwa versuchen, eines davon dazu zu bestimmen, alle anderen Geräte zu testen", meint Böhme. Auch die Sicherheit von Machine-Learning-Algorithmen, also der künstlichen Intelligenz, spielt eine immer größere Rolle in der Gesellschaft. „Diese Systeme funktionieren nach völlig anderen Prinzipien als klassische Computerprogramme", sagt der Forscher. „Aber leider haben wir noch überhaupt keine Techniken, um sicherzustellen, dass etwa ein KI-Assistent auch wirklich das macht, was er soll." Das zu ändern betrachtet Böhme als wichtige Aufgabe, der er sich in Zukunft gemeinsam mit seinem Team vermehrt widmen möchte. *www.mpg.de/podcasts/sicherheit*



FOTO: FRANK VINKEN FÜR MPG

Wettlauf mit den Angreifern: Marcel Böhme und sein Team arbeiten daran, das Internet der Dinge und künstliche Intelligenz sicher zu machen. Mit Kirandeep Kaur diskutiert er eine neue Idee, um Schlupftüren für Cyberkriminelle zu schließen.

### GLOSSAR

*FUZZING*
(abgeleitet von *fuzzy*, englisch für „unscharf") bezeichnet automatisierte Testverfahren, bei denen Software mit zufälligen Eingaben auf Sicherheitslücken überprüft wird. Es wird unterschieden zwischen dem rein zufälligen Blackbox Fuzzing, dem Whitebox Fuzzing mit zusätzlichen Modellanalysen und dem Greybox Fuzzing, bei dem ein Programm systematisch mit massenhaften, zufällig gewählten Eingaben getestet wird.

*STATISCHE SOFTWAREANALYSE*
versucht mathematisch zu beweisen, dass keine mögliche Eingabe zu einem kritischen Fehler im Programm führen kann.