1 Sicherheit in einer Welt der Webanwendungen

Um die Grundlagen für die technischen Erläuterungen in diesem Buch zu legen, sind zunächst einmal zwei Fragen zu klären: Was soll eigentlich durch Sicherheitstechniken erreicht werden? Und warum erfordern ausgerechnet Webanwendungen bei diesem gut erforschten Thema eine besondere Behandlung? Legen wir also los!

1.1 Informationssicherheit kurz und bündig

Oberflächlich betrachtet scheint das Gebiet der Informationssicherheit ein ausgereifter, wohldefinierter und abgeschlossener Zweig der Informatik zu sein. Die entsprechenden Experten versichern uns eifrig, wie wichtig ihr Fachgebiet sei, und weisen uns auf umfangreiche Kataloge sorgfältig dokumentierter Sicherheitslücken hin, die durch die Bank Entwicklern mit geringem Sicherheitsbewusstsein in die Schuhe geschoben werden. Andere Fachleute heben währenddessen hervor, dass all diese Probleme vermieden werden könnten, wenn man sich nur an die brandaktuellste Sicherheitsmethodik hielte. Im kommerziellen Umfeld dieses Themas gedeiht gleichzeitig eine Branche, die die verschiedensten, unverbindlichen Sicherheitsversprechen abgibt, sowohl gegenüber einfachen Computerbenutzern als auch gegenüber großen internationalen Firmen.

In Wirklichkeit haben wir es aber seit mehreren Jahrzehnten nicht geschafft, auch nur rudimentär verwendbare Bezugssysteme zu erstellen, mit denen man die Sicherheit moderner Software untersuchen und bewerten könnte. Trotz mehrerer brillanter Aufsätze und kleinerer Experimente können wir immer noch keine Erfolgsgeschichten aus der Praxis vorweisen. Der Schwerpunkt liegt fast ausschließlich auf reaktiven, sekundären Sicherheitsmaßnahmen (z.B. Vulnerability Management, Erkennung von Malware und Angriffen, Verwendung von Sandboxes) oder besteht manchmal einfach darin, die Sicherheitsmängel im Code anderer Leute aufzuzeigen. Das entmutigende und eifersüchtig gehütete Geheimnis lautet, dass wir nur wenig dazu beitragen, anderen bei der Entwicklung sicherer Systeme zu helfen. Und das moderne Web bildet da keine Ausnahme.

Hinweis

Ein oft unbeachtetes Problem vieler Analyse- und Schutzwerkzeuge ist die Frage des Kontextes. Ohne Kenntnis über den Kontext der verwendeten Daten kann kein effektiver Schutz stattfinden. In einem HTML-Element zum Beispiel wird Plaintext als harmlos wahrgenommen – aber nur wenn es sich nicht um Style- oder Script-Elemente handelt. Bilder stellen selten eine große Sicherheitsbedrohung dar – aber nur wenn in der URL keine sensitiven Daten transportiert werden und der »Referrer« nicht »leaken« darf. Ohne komplexes Metawissen sind viele Tools aufgeschmissen und können die eigenen Sicherheitsversprechen nur schwer bis gar nicht einhalten.

Sehen wir uns einige der verheißungsvollsten Ansätze zur Informationssicherheit an und versuchen wir herauszufinden, warum sie bis jetzt noch keine Wirkung gezeigt haben.

1.1.1 Liebäugeln mit formalen Lösungen

Der wohl naheliegendste Weg, um sichere Programme zu erstellen, besteht darin, algorithmisch zu beweisen, dass sie sich wie gewünscht verhalten. Das ist eine einfache Annahme, die instinktiv durchaus realistisch klingt. Warum hat dieser Ansatz dann unter dem Strich nicht viel eingebracht?

Sehen wir uns als Erstes das Wörtchen *sicher* genauer an: Was sollte es ursprünglich bedeuten? Sicherheit scheint ein allgemein verständlicher Begriff zu sein, aber in der Welt der Computer entzieht er sich jeglicher nützlichen Definition. Natürlich könnten wir das Problem auf griffige, aber nicht sehr hilfreiche Weise umformulieren. Es ist allerdings offensichtlich, dass wir in Schwierigkeiten geraten, wenn eine der Definitionen, die viele Fachleute in der Praxis[1] verwenden, wie folgt lautet:

> *Ein System ist sicher, wenn es sich genau in der vorgesehenen Weise verhält – und nichts anderes tut.*

Diese Definition ist nett und umreißt vage ein abstraktes Ziel, sagt aber wenig darüber aus, wie wir es erreichen sollen. Es geht hier um Informatik, doch ist die Aussage ungefähr so konkret wie in einem Gedicht von Victor Hugo:

> *Liebe ist ein Teil der Seele selbst, und sie ist von derselben Natur wie der himmlische Atem der Atmosphäre im Paradies.*

Sie könnten einwenden, dass man Praktiker nicht nach differenzierten Definitionen fragen soll, aber wenn Sie dieselbe Frage Akademikern stellen, erhalten Sie im Großen und Ganzen die gleichen Antworten. Die folgende akademische Defi-

1. Das ursprüngliche Zitat wird Ivan Arce zugeschrieben, einem renommierten Bughunter. Seit 2000 wurde es von Crispin Cowan, Michael Howard, Anton Chuvakin und Dutzenden anderer Sicherheitsexperten verwendet.

nition geht beispielsweise auf das Sicherheitsmodell Bell-La Padula zurück, das in den 1960er-Jahren veröffentlicht wurde. (Das war einer von ungefähr einem Dutzend Versuchen, Anforderungen für Sicherheitssysteme zu formalisieren, in diesem Fall für einen endlichen Zustandsautomaten [1]. Es ist auch einer der bemerkenswertesten Definitionsversuche.)

> *Ein System ist genau dann sicher, wenn es in einem sicheren Zustand beginnt und keinen unsicheren Zustand annehmen kann.*

Definitionen dieser Art sind natürlich grundsätzlich wahr und können als Ausgangspunkt für Dissertationen und sogar staatlich bezuschusste Projekten dienen. Modelle, die auf dieser Grundlage erstellt werden, sind für die allgemeine Softwareentwicklung in der Praxis jedoch nahezu nutzlos, und das aus folgenden drei Gründen:

- **Es gibt keine Möglichkeit, das wünschenswerte Verhalten eines ausreichend komplexen Computersystems zu definieren.**
  Keine einzelne Instanz kann definieren, was die »vorgesehene Weise« oder ein »sicherer Zustand« eines Betriebssystems oder eines Webbrowsers ist. Die Interessen der Benutzer, der Systembetreiber, der Datenlieferanten, der Geschäftsprozessverantwortlichen und der Software- und Hardwarehersteller weichen stark voneinander ab und ändern sich rasch – falls die Beteiligten überhaupt willens und in der Lage sind, ihre Interessen ehrlich und deutlich offenzulegen. Und das ist noch nicht alles, denn Soziologie und Spieltheorie legen den Verdacht nahe, dass eine einfache Addition all dieser Einzelinteressen nicht zu einem vorteilhaften Ergebnis führen wird. Dieses Dilemma, das sogenannte Allmende-Problem oder die »Tragik der Allmende«, ist ein zentraler Punkt vieler Diskussionen über die Zukunft des Internets.
- **Wunschdenken lässt sich nicht automatisch auf formale Bedingungen abbilden.**
  Selbst wenn wir eine perfekte, abstrakte Übereinkunft darüber erzielen können, wie sich ein System in einer Teilmenge der Fälle verhalten soll, ist es so gut wie unmöglich, diese Erwartungen zu einem Satz von zulässigen Eingaben, Programmzuständen und Zustandsübergängen zu formalisieren, was die Voraussetzung für jede Art formaler Analyse ist. Unmittelbar einleuchtende Aussagen wie: »Ich will nicht, dass meine E-Mails von anderen gelesen werden können« lassen sich schlichtweg nicht besonders gut in mathematische Modelle übersetzen. Es gibt verschiedene exotische Ansätze, um solche vagen Anforderungen zumindest teilweise zu formalisieren, aber sie schränken die Prozesse der Softwareentwicklung stark ein und führen oft zu Regeln und Modellen, die viel komplizierter sind als die Algorithmen, die damit zu validieren sind. Außerdem müssen sie selbst ihre eigene Richtigkeit unter Beweis stellen, und diese Kette können Sie *ad infinitum* fortsetzen.

- **Das Softwareverhalten lässt sich nur sehr schwer schlüssig analysieren.**
  In komplexen Fällen aus der Praxis hat es noch niemand überzeugend geschafft, mit einer statischen Analyse von Computerprogrammen zu beweisen, dass sich die betreffenden Anwendungen immer gemäß der ausführlichen Spezifikation verhalten werden. (Bei sehr eingeschränkten Einstellungen oder sehr engen Zielsetzungen ist damit jedoch ein begrenzter Erfolg möglich.) Viele Fälle sind in der Praxis wahrscheinlich unmöglich zu lösen (aufgrund der rechentechnischen Komplexität), und viele sind aufgrund eines Halteproblems möglicherweise sogar gar nicht entscheidbar.[2]

Noch deprimierender als die Ungenauigkeit und Nutzlosigkeit dieser frühen Definitionen ist vielleicht die Tatsache, dass nach Jahrzehnten immer noch kaum Fortschritte auf dem Weg zu etwas Besserem erzielt wurden. Ein 2001 vom US-amerikanischen Naval Research Laboratory veröffentlichter wissenschaftlicher Artikel geht auf diese älteren Arbeiten zurück und kommt zu einer noch viel informelleren Definition – die ausdrücklich die Beschränktheit und Unvollständigkeit von Informationssicherheit betont [2].

> *Ein System ist sicher, wenn es die von ihm verarbeiteten Informationen adäquat schützt gegen unautorisierte Offenlegung, unautorisierte Veränderung und unautorisierte Zurückhaltung (auch Denial of Service genannt). Wir sagen hier »adäquat«, da kein System in der Praxis diese Ziele ohne Einschränkungen erreicht. Sicherheit ist von Natur aus relativ.*

Diese Arbeit liefert auch eine Bewertung früherer Ansätze und der unannehmbaren Kompromisse, die gemacht wurden, um die theoretische Reinheit dieser Modelle zu bewahren:

> *Auf der einen Seite hat die Erfahrung gezeigt, dass die Axiome des Bell-La-Padula-Modells übermäßig restriktiv sind: Sie verbieten Operationen, die die Benutzer in praktischen Anwendungen benötigen. Auf der anderen Seite sind die Elemente, denen vertraut wird, um die Sicherheit durchzusetzen, nicht restriktiv genug ... Infolgedessen mussten Entwickler Ad-hoc-Spezifikationen für das gewünschte Verhalten von vertrauenswürdigen Prozessen in jedem einzelnen System aufstellen.*

Trotz der großen Anzahl eleganter und miteinander im Wettstreit stehender Modelle scheinen unter dem Strich alle Versuche, die Sicherheit von Software auf der Grundlage von Algorithmen zu ermitteln und zu bewerten, zum Scheitern verurteilt zu sein. Somit bleibt den Entwicklern und den Sicherheitsexperten keine Methode mehr, um maßgebliche, vorausschauende Aussagen über die Qualität von Code zu machen. Welche anderen Möglichkeiten gibt es also noch?

---

2. 1936 zeigte Alan Turing (mit einer leicht abweichenden Formulierung), dass es nicht möglich ist, einen Algorithmus zu entwerfen, der allgemein das Ergebnis anderer Algorithmen entscheiden kann. Natürlich sind einige Algorithmen sehr gut durch fallspezifische Tests entscheidbar, aber nicht alle.

1.1.2 Risikomanagement und IT-Sicherheit

Angesichts des Mangels an formalen Sicherheiten und tauglichen Messgrößen sowie der erschreckenden Sicherheitsmängel in Softwaresystemen, die für die moderne Gesellschaft von zentraler Bedeutung sind, gedeiht das Geschäft mit einem anderen einprägsamen Begriff: *Risikomanagement*.

Das Prinzip des Risikomanagements, das in der Versicherungsbranche erfolgreich (und in der Finanzwelt mit etwas weniger Erfolg) angewandt wird, ist einfach und lautet: Die Besitzer von Systemen müssen mit all jenen Schwachstellen leben, die sich nicht auf kostengünstige Weise ausmerzen lassen. Und sie sollten die Aufwände für Sicherheitsmaßnahmen ins Verhältnis zum Risiko setzen, und zwar nach folgender Formel:

Risiko = Wahrscheinlichkeit eines Ereignisses × maximaler Verlust

Wenn der Ausfall eines unbedeutenden Arbeitsplatzrechners das Unternehmen im Jahr nicht mehr als 1000 € an verlorener Produktivität kostet, sollte die Organisation nach dieser Doktrin einfach nur das Budget für diesen Verlust vorsehen, anstatt 100.000 € für zusätzliche Sicherheitsmaßnahmen oder Notfall- und Überwachungspläne zur Verhinderung des Verlustes auszugeben. Nach der Lehre des Risikomanagements ist es besser, das Geld woanders zu investieren, nämlich in die Isolierung, Absicherung und Überwachung des geschäftswichtigen Mainframe-Computers, der die Rechnungen für sämtliche Kunden erstellt.

Natürlich ist es klug, die Sicherheitsanstrengungen nach Prioritäten zu ordnen. Wenn Risikomanagement aber einfach nur nach Zahlenwerten durchgeführt wird, hilft uns das jedoch sehr wenig, um Praxisprobleme zu verstehen, einzugrenzen und zu lösen. Stattdessen führt es zu dem gefährlichen Trugschluss, dass strukturierte unangemessene Zustände fast genauso gut seien wie angemessene Zustände und dass unterfinanzierte Sicherheit plus Risikomanagement fast genauso gut wäre wie eine ausreichend finanzierte Sicherheit.

Aber auch damit kommen Sie nicht weiter.

- **In vernetzten Systemen sind Verluste nicht isoliert und nicht auf einen einzelnen Budgetposten begrenzt.**
  Strenges Risikomanagement geht davon aus, dass man die typischen und maximalen Kosten für den Missbrauch einer Ressource abschätzen kann. Das ist jedoch nur möglich, wenn man die Tatsache ignoriert, dass die spektakulärsten Sicherheitsverletzungen – beispielsweise die Angriffe auf TJX[3] oder

3. Im Jahr 2006 griffen mehrere Eindringlinge, angeblich angeführt von Albert Gonzalez, ein unsicheres drahtloses Netzwerk in der Filiale eines Einzelhandelsriesen an und konnten darüber schließlich ins Firmennetzwerk des Unternehmens vorstoßen. Sie kopierten die Kreditkartendaten von etwa 46 Millionen Kunden sowie die Sozialversicherungsnummern, Postanschriften usw. von weiteren 450.000. Elf Personen wurden in Verbindung mit diesem Angriff angeklagt, wobei eine von ihnen Selbstmord beging.

Microsoft[4] – an relativ unwichtigen und vernachlässigbaren Eintrittspunkten begannen. Die Eindringlinge bauten ihre ersten Erfolge weiter aus und umgingen dabei alle oberflächlichen Abschottungen des Netzwerks, was schließlich zum fast vollständigen Missbrauch einer kritischen Infrastruktur führte. Bei der üblichen Zahlenspielerei im Risikomanagement wird dem Eintrittspunkt nur ein geringes Gewicht beigemessen, da er im Vergleich zu anderen Knoten nur einen geringen Wert darstellt. Ebenso wird der interne Eskalationspfad zu immer sensibleren Ressourcen heruntergespielt, da die Wahrscheinlichkeit für seinen Missbrauch gering ist. Beides zu ignorieren ist jedoch eine hochexplosive Mischung.

- **Die nicht monetären Kosten eines Eindringens lassen sich nur schwer gegen den Wert eines gesunden Systems abwägen.**
  Gegen den Verlust des Benutzervertrauens, eine Unterbrechung der Geschäfte, mögliche Klagen und staatliche Untersuchungen kann man sich nicht sinnvoll versichern. Solche Auswirkungen aber können zumindest prinzipiell über Sein oder Nichtsein eines Unternehmens oder sogar ganzer Branchen entscheiden. Jede oberflächliche Bewertung solcher Folgen ist nahezu reine Spekulation.
- **Vorhandene Daten sind möglicherweise nicht repräsentativ für zukünftige Risiken.**
  Anders als die Unfallgegner bei einem kleinen Blechschaden stellen sich die Angreifer nicht vor, um ihre Einbrüche zu melden, und sie dokumentieren den angerichteten Schaden auch nicht. Sofern der Einbruch nicht auf schmerzhafte Weise offensichtlich ist (weil der Angreifer schlampig vorgegangen ist oder mit seiner Tat eine Störaktion verüben wollte), bleibt er oft unbemerkt. Selbst wenn branchenweit freiwillig bereitgestellte Daten von Betroffenen vorliegen, gibt es einfach keine zuverlässige Möglichkeit, um zu beurteilen, wie vollständig sie sind oder wie viel zusätzliches Risiko die derzeitigen Geschäftspraktiken hervorrufen.
- **Statistische Prognosen können einzelne Ergebnisse nicht zuverlässig vorhersagen.**
  Auch wenn die Stadtbevölkerung im statistischen Mittel häufiger vom Blitz getroffen als von Bären gefressen wird, heißt das nicht, dass Sie sich einen Blitzableiter an den Hut montieren und dann in Honig baden sollten. Die Wahrscheinlichkeit dafür, dass mit einer bestimmten Komponente Missbrauch getrieben wird, ist für den Einzelfall größtenteils irrelevant: Dass es zu

4. Die offiziell nicht veröffentlichte Microsoft-Präsentation mit dem nichtssagenden Titel *Threats Against and Protection of Microsoft's Internal Network* (»Bedrohungen und Schutz des internen Netzwerks von Microsoft«) beschreibt einen Angriff aus dem Jahr 2003. Ausgangspunkt war der gekaperte Heimarbeitsplatz eines Ingenieurs, der langlebige VPN-Verbindungen mit dem Unternehmensnetzwerk unterhielt. Darauf folgten methodische Versuche der Angreifer, noch mehr zu erreichen, die schließlich darin gipfelten, dass sie Zugang auf interne Code-Repositories bekamen und Daten daraus abzapften. Die Täter sind bis heute, zumindest in der Öffentlichkeit, unbekannt.

Sicherheitsverletzungen kommt, ist so gut wie sicher, und von den Tausenden offen liegender nicht trivialer Ressourcen kann jeder Dienst als Angriffspfad dienen. Kein einzelner Dienst aber wird eine solche Menge an Ereignissen erfahren, dass damit eine sinnvolle Vorhersage im Rahmen eines einzigen Unternehmens möglich wäre.

1.1.3 Erleuchtung durch Taxonomie

Die beiden oben vorgestellten Denkweisen haben eines gemeinsam: Beide gehen davon aus, dass es möglich ist, Sicherheit in Form einer Reihe berechenbarer Ziele zu definieren, und dass wir über die resultierende einheitliche Theorie eines Sicherheitssystems bzw. über ein Modell akzeptabler Risiken eine Menge optimaler und konkreter Aktionen für ein perfektes Anwendungsdesign erhalten.

Manche Praktiker predigen jedoch den gegenteiligen Ansatz, der weniger den Geistes- als vielmehr den Naturwissenschaften entsprungen ist. Wie ein Charles Darwin des Informationszeitalters argumentieren sie, dass es durch die Erfassung ausreichender Mengen konkreter experimenteller Daten möglich ist, immer anspruchsvollere Gesetze zu erkennen, zu rekonstruieren und zu dokumentieren, um schließlich zu einem einheitlichen Modell der sicheren Computernutzung zu kommen.

Diese Weltsicht hat Projekte hervorgebracht wie das vom US-Ministerium für Heimatschutz finanzierte CWE (Common Weakness Enumeration). Mit den eigenen Worten der Organisation besteht ihr Ziel darin, eine einheitliche »Theorie der Schwachstellen« zu entwickeln, »die Forschung, Modellierung und Klassifizierung von Softwaremängeln zu verbessern« und »eine gemeinsame Sprache bereitzustellen für die Diskussion über die Ursachen von Sicherheitsmängeln in Software, die Suche danach und den Umgang damit«. Ein typisches, herrlich verschnörkeltes Beispiel der resultierenden Taxonomie sieht wie folgt aus:

Ungenügende Anwendung von Nachrichten- oder Datenstrukturen
Versäumnis, die Daten für eine andere Ebene zu bereinigen
Ungenügende Steuerung der Ressourcenbezeichner
Ungenügende Filterung der Namen von Dateien und anderen Ressourcen für ausführbare Inhalte

Heute enthält das CWE-Wörterbuch über 800 Einträge, von denen die meisten so diskussionsfördernd und hilfreich sind wie der hier zitierte.

Eine abweichende naturalistische Denkweise zeigt sich in Projekten wie dem Common Vulnerability Scoring System (CVSS), einer von Unternehmen unterstützten gemeinsamen Maßnahme, die bekannte Sicherheitsprobleme in Form grundlegender, maschinenlesbarer Parameter genau quantifizieren soll. Ein tatsächliches Beispiel eines solchen Schwachstellenbezeichners sieht wie folgt aus:

```
AV:LN / AC:L / Au:M / C:C / I:N / A:P / E:F / RL:T / RC:UR /
CDP:MH / TD:H / CR:M / IR:L / AR:M
```

Von Organisationen und Forschern wird erwartet, diesen 14-dimensionalen Vektor auf eine sorgfältig ausgewählte, anwendungsspezifische Weise zu transformieren und damit zu einer objektiven, überprüfbaren, numerischen Schlussfolgerung über die Signifikanz eines zugrunde liegenden Bugs (z.B. »42«) zu kommen, ohne dass es nötig ist, die Natur von Sicherheitsmängeln auf subjektivere Weise zu bewerten.

Ja, ich mache mich ein bisschen über diese Projekte lustig, aber ich möchte diese Maßnahmen auch nicht herabsetzen. CWE, CVSS und ähnliche Projekte dienen hehren Zielen. Beispielsweise sollen sie die Sicherheitsprozesse großer Organisationen in einen leichter zu handhabenden Rahmen rücken. Allerdings hat noch nichts davon zu einer großen Theorie sicherer Software geführt, und ich bezweifle auch, dass ein solches System in Sicht ist.

1.1.4 Auf dem Weg zu praktischen Ansätzen

Zurzeit deuten alle Zeichen darauf hin, dass Sicherheit zum Großteil ein nicht algorithmisches Problem ist. Die Branche tut sich naturgemäß schwer damit, dies offen anzuerkennen, denn es bedeutet, dass man kein Allheilmittel verkünden (und kommerziell nutzen) kann. Wenn es hart auf hart kommt, greifen jedoch fast alle, die mit Sicherheit zu tun haben, auf eine Reihe rudimentärer, empirischer Rezepte zurück. Diese Rezepte sind zwar größtenteils unvereinbar mit vielen Modellen der Betriebswirtschaft, aber sie haben bis jetzt gut funktioniert. Es handelt sich dabei um folgende:

- **Aus Fehlern lernen (bevorzugt aus denen anderer Leute)**
  Systeme sollten so entworfen werden, dass die bekannten Klassen von Bugs ausgeschlossen sind. Da es keine automatischen Lösungen gibt (nicht einmal elegante), lässt sich dieses Ziel am besten durch ständige Anleitung bei Design und Entwicklung erreichen. Dabei erhalten die Entwickler die Kenntnisse darüber, was schiefgehen kann, und die Werkzeuge, um fehleranfällige Aufgaben auf die einfachste mögliche Weise zu lösen.
- **Werkzeuge zum Erkennen und Lösen von Problemen entwickeln**
  Sicherheits-mängel haben gewöhnlich keine offensichtlichen Nebenwirkungen, bis sie von böswilligen Personen erkannt werden. Das ist jedoch eine ziemlich teure Art von Rückmeldung. Als Gegenmaßnahme erstellen wir Qualitätssicherungswerkzeuge (Quality Assurance, QA), um Implementierungen zu überprüfen, und führen regelmäßig Tests aus, um gelegentliche Fehler (oder systematische Fehler bei der Entwicklung) aufzuspüren.

- **Überall den Missbrauch wittern**
  Die Geschichte lehrt uns, dass trotz unserer besten Abwehrmaßnahmen schwerwiegende Vorfälle geschehen. Es ist wichtig, eine angemessene Trennung der Komponenten, eine Zugriffssteuerung, Datenredundanz, Überwachung und Verfahren für den Ernstfall vorzusehen. Damit können die Besitzer des Dienstes reagieren, bevor sich ein kleiner Schluckauf zu einer Katastrophe biblischen Ausmaßes entwickelt.

In jedem Fall sind eine große Portion Geduld, Kreativität und technische Kenntnisse von all jenen erforderlich, die mit der Informationssicherheit zu tun haben.

Naturgemäß werden selbst einfache Regeln, die schon der gesunde Menschenverstand lehrt, – vor allem die grundlegende Exaktheit des Entwicklungsvorgangs – in Schlagwörter gekleidet, mit Abkürzungen verziert (etwa CIA für *Confidentiality, Integrity, Availability*, also »Vertraulichkeit, Integrität, Verfügbarkeit«) und dann als »Methodik« bezeichnet. Häufig sind solche Methodiken nur leicht verschleierte Versuche, einen der frustrierendsten Fehlschläge der Sicherheitsbranche zu einer weiteren Erfolgsstory umzumünzen und leichtgläubigen Kunden ein weiteres Wundermittel oder noch eine Zertifizierung zu verkaufen. Trotz aller gegenteiligen Behauptungen sind solche Produkte jedoch kein Ersatz für Know-how und technisches Können – zumindest nicht in der heutigen Zeit.

Auf jeden Fall werde ich im weiteren Verlauf dieses Buches nicht versuchen, eines der zuvor erwähnten großen philosophischen Bezugssysteme zu etablieren oder wiederzuverwenden. Stattdessen greife ich auf eine gesunde Dosis Anti-Intellektualismus zurück. Ich untersuche die offen liegende Oberfläche moderner Browser und erörtere, wie die verfügbaren Werkzeuge auf sichere Weise eingesetzt werden können, welche Teile des Web gewöhnlich falsch verstanden werden und wie sich der Schaden im Ernstfall begrenzen lässt.

Das ist so ziemlich der beste Ansatz zur sicheren Softwareentwicklung, den ich kenne.

1.2 Eine kurze Geschichte des Web

Das Web leidet an einer erstaunlichen Anzahl und einer bemerkenswerten Vielfalt von Sicherheitsproblemen. Gewiss gehen manche dieser Probleme auf einmalige Pannen in bestimmten Client- oder Serverimplementierungen zurück. Viele davon haben wir aber auch launenhaften und oft willkürlichen Designentscheidungen darüber zu verdanken, wie die wichtigen Mechanismen funktionieren und im Browser verzahnt werden.

Unser Web steht auf tönernen Füßen – aber warum? Vielleicht liegt es einfach an einer gewissen Kurzsichtigkeit: Wer konnte damals in den Tagen der Unschuld schon die Gefahren der heutigen Netzwerke und die wirtschaftliche Verlockung von Cyber-Angriffen vorhersagen?

Diese Erklärung mag zwar für wirklich altehrwürdige Mechanismen wie SMTP oder DNS gelten, aber leider nicht für das Web, das relativ jung ist und in einer Umgebung Form annahm, die sich nicht stark von der heutigen unterscheidet. Die Lösung des Rätsels liegt vermutlich in der chaotischen und ungewöhnlichen Art und Weise, in der sich die zugehörigen Technologien entwickelt haben.

Entschuldigen Sie daher bitte, dass ich schon wieder abschweife und zu den Wurzeln zurückkehre. Die Vorgeschichte des Web ist ziemlich banal, aber trotzdem einen genaueren Blick wert.

1.2.1 Geschichten aus der Steinzeit: 1945 bis 1994

Computerhistoriker erwähnen häufig ein hypothetisches Gerät von Schreibtischgröße namens Memex, einen der ersten Fossilienfunde, den Vannevar Bush 1945 postulierte [3]. Memex sollte es möglich machen, dokumentübergreifende Querverweise in Mikrofilm zu erstellen, zu kommentieren und zu verfolgen. Die dazu verwendete Technik ähnelte entfernt den modernen Browserlesezeichen und Hyperlinks. Bush wagte die kühne Vorhersage, dass diese einfache Fähigkeit das Gebiet der Wissensverwaltung und des Datenabrufs revolutionieren würde. (Amüsanterweise machte man sich bis in die 1990er-Jahre gelegentlich noch über diese Behauptung lustig und bezeichnete sie als ungebildet und naiv.) Nutzbare Umsetzungen dieses Designs waren in der damaligen Zeit jedoch nicht möglich, sodass sich außer Zukunftsvisionen nichts entwickelte, bis die Transistorcomputer die Bühne betraten.

Der nächste Meilenstein war die Entwicklung der Sprache GML (Generalized Markup Language) durch IBM in den 1960er-Jahren. Dadurch wurde es möglich, Dokumente mit maschinenlesbaren Zusätzen zu versehen, die die Funktion jedes einzelnen Textblocks angaben. Im Grunde genommen besagten diese Zusätze: »Dies ist eine Überschrift«, »Dies ist eine nummerierte Liste« usw. Im Verlauf der nächsten 20 Jahre wurde GML (ursprünglich nur von einer Handvoll IBM-Texteditoren und sperrigen Mainframe-Computern verwendet) zur Grundlage der Standard Generalized Markup Language (SGML), einer universeller einsetzbaren und flexibleren Sprache, in der die unbeholfene Syntax mit Doppelpunkten und Punkten durch die vertraute Schreibweise mit spitzen Klammern ersetzt wurde.

Während sich GML zu SGML weiterentwickelte, wurden Computer leistungsfähiger und benutzerfreundlicher. Verschiedene Forscher begannen mit Bushs Querverweiskonzept zu experimentieren und es auf die Speicherung und den Abruf von Daten in Computersystemen anzuwenden. Dadurch wollten Sie ermitteln, ob es möglich wäre, große Mengen von Dokumenten mithilfe einer Art von Schlüssel über Querverweise zu verknüpfen. Experimentierfreudige Unternehmen und Universitäten führten Pionierprojekte wie ENQUIRE, NLS und Xanadu durch, aber die meisten konnten keine bleibenden Ergebnisse erzielen. Die Kritiken an den verschiedenen Projekten betrafen die begrenzte praktische Nutzbarkeit, die extreme Kompliziertheit und die schlechte Skalierbarkeit.

Am Ende des Jahrzehnts hatten die beiden Forscher Tim Berners-Lee und Dan Connolly mit der Arbeit an einem neuen Ansatz für domänenübergreifende Querverweise begonnen – mit dem Schwerpunkt auf Einfachheit. Sie starteten das Projekt mit der Entwicklung der Sprache HTML (HyperText Markup Language), einem einfachen Abkömmling von SGML, der eigens dazu entworfen war, Dokumente mit Hyperlinks und einer grundlegenden Formatierung zu versehen. Die Arbeit an HTML setzten sie mit der Entwicklung von HTTP (HyperText Transfer Protocol) fort, einem extrem einfachen, auf den Zugriff von HTML-Ressourcen zugeschnittenen Verfahren, das die vorhandenen Konzepte von IP-Adressen, Domänennamen und Dateipfaden nutzte. Ihre Arbeit gipfelte irgendwann zwischen 1991 und 1993 im World Wide Web (siehe Abbildung 1–1), einem rudimentären Browser, der HTML-Text darstellte. Damit konnten Benutzer die resultierenden Daten auf dem Bildschirm sehen und mit einem Mausklick von einer Seite zur anderen gelangen.

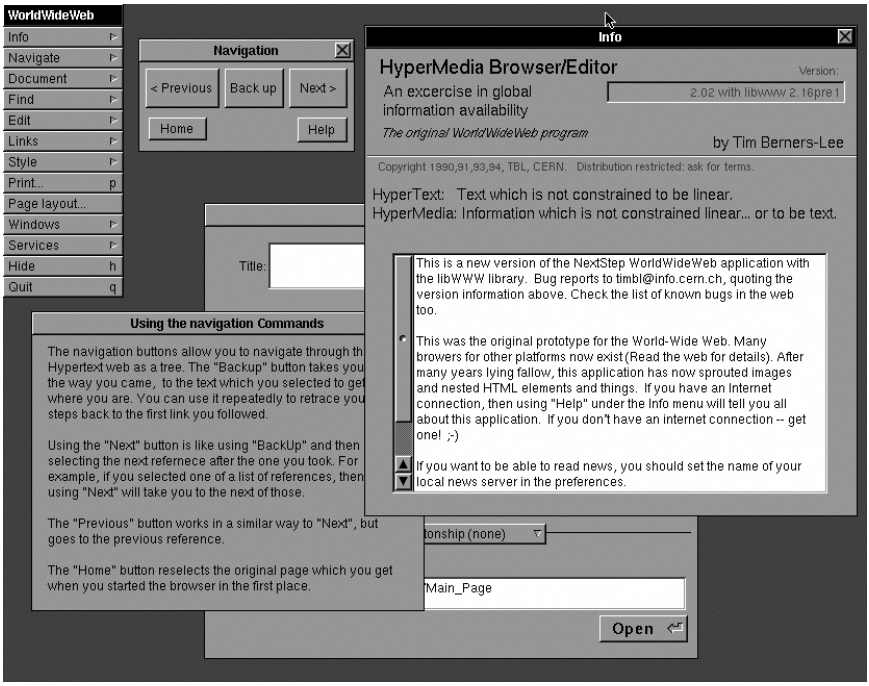

Abb. 1–1 *Das World Wide Web von Tim Berners-Lee*

Für viele muss das Design von HTTP und HTML wie ein erheblicher Rückschritt im Vergleich zu den hochfliegenden Zielen der Konkurrenzprojekte gewirkt haben. Schließlich brüsteten sich viele der früheren Arbeiten mit Merkmalen wie Datenbankintegration, Sicherheit und digitaler Rechteverwaltung oder kooperativem Publizieren. Selbst Tim Berners-Lees eigenes Projekt ENQUIRE erschien viel ambitionierter als seine Arbeit am Web. Trotzdem wurde das unscheinbare

WWW-Projekt zu einem Überraschungserfolg – aufgrund seiner geringen Einstiegshürden, der unmittelbaren Gebrauchsfähigkeit und der unbegrenzten Skalierbarkeit (was zeitlich zufällig mit dem Aufkommen leistungsfähigerer und erschwinglicher Computer sowie der Erweiterung des Internets zusammenfiel).

Ja, ich gebe zu, ein »Erfolg« war es nur nach den Maßstäben der 1990er-Jahre. Schon bald gab es Dutzende von Webservern im Internet. Bis 1993 nahm der HTTP-Datenverkehr 0,1% der gesamten Bandbreite im Backbone-Netzwerk der National Science Foundation ein. Im selben Jahr kam auch Mosaic auf, der erste halbwegs populäre und anspruchsvolle Webbrowser, entwickelt von der University of Illinois. Mosaic erweiterte den ursprünglichen World-Wide-Web-Code um Funktionen wie die Einbettung von Bildern in HTML-Dokumenten und die Übermittlung von Benutzerdaten in Formularen. Damit war der Weg für die heutigen interaktiven Multimedia-Anwendungen frei.

Mosaic machte das Surfen angenehmer und förderte die Akzeptanz des Web durch den Endverbraucher. Mitte der 1990er-Jahre diente er außerdem als Grundlage für zwei andere Browser: Mosaic Netscape (der später in Netscape Navigator umbenannt wurde) und Spyglass Mosaic (schließlich von Microsoft aufgekauft und in Internet Explorer umbenannt). Es kam auch eine Handvoll Browser auf, die nicht auf der Mosaic-Engine beruhten, darunter Opera und verschiedene Textbrowser (wie Lynx und w3m). Und wenig später folgten die ersten Suchmaschinen, Onlinezeitungen und Dating-Websites.

1.2.2 Die ersten Browserkriege: 1995 bis 1999

Mitte der 1990er-Jahre war klar, dass das Web Bestand haben würde und dass die Benutzer bereit waren, viele ältere Technologien zugunsten des neuen Rivalen aufzugeben. Microsoft war groß in Desktop-Software, hatte sich bis dato aber schwergetan, auf den Internetzug aufzuspringen. Jetzt jedoch wurde das Unternehmen unruhig und begann erhebliche Entwicklungsressourcen in einen eigenen Browser zu stecken und ihn 1996 schließlich sogar mit dem Betriebssystem Windows zu bündeln.[5] Die Maßnahmen von Microsoft leiteten eine Zeit ein, die umgangssprachlich als »die Browserkriege« bezeichnet wird.

Das daraus hervorgehende Wettrüsten der Browserhersteller zeichnete sich durch eine bemerkenswert rasche Entwicklung und Bereitstellung von neuen Funktionen in den konkurrierenden Produkten aus. Dieser Trend machte häufig

5. Diese Entscheidung erwies sich interessanterweise als sehr umstritten. Einerseits kann man sagen, dass Microsoft damit die Popularität des Internets stark gefördert hat. Andererseits hat das Unternehmen dadurch jedoch auch die Position von Konkurrenzbrowsern unterminiert, was als wettbewerbsschädigend angesehen werden kann. Diese Vorgehensweise führte schließlich zu einer Reihe langwieriger Rechtsstreitigkeiten über die mögliche Monopolbildung des Unternehmens, die in Europa schließlich dafür sorgten, dass in Windows auch die Installation anderer Browser angeboten werden muss.

alle Versuche zunichte, den neu hinzugefügten Code zu standardisieren oder zumindest sauber zu dokumentieren. Die Änderungen am HTML-Kern umfassten teilweise sinnlose Ergänzungen (wie die von Netscape erfundene Möglichkeit, Text blinken zu lassen, die zur Zielscheibe des Spotts und zu einem untrüglichen Erkennungszeichen für schlechtes Webdesign wurde). Es gab aber auch hervorragende Entwicklungen, etwa die Möglichkeit, die Schriftart zu ändern oder externe Dokumente in sogenannten Frames einzubetten. Die Hersteller lieferten ihre Produkte mit eingebetteten Programmiersprachen wie JavaScript und Visual Basic aus, mit Plug-ins zur Ausführung plattformunabhängiger Java- oder Flash-Applets auf dem Benutzercomputer sowie mit nützlichen, aber heiklen HTTP-Erweiterungen wie Cookies. Kompatibilität war nur oberflächlich und in begrenztem Maße gewährleistet und manchmal auch noch durch Patente und Marken behindert.[6]

Als das Web größer und vielfältiger wurde, erfasste die Browser-Engines eine schleichende Krankheit, die unter dem Deckmäntelchen der Fehlertoleranz daherkam. Zu Anfang erschien die Argumentation völlig sinnvoll: Wenn Browser A eine schlecht entworfene, defekte Seite anzeigen konnte, Browser B sich aber (aus welchem Grund auch immer) weigerte, das zu tun, würden die Benutzer das Versagen von Browser B unvermeidlich als Fehler des Produkts ansehen und in Scharen zu dem vermeintlich fähigeren Client überlaufen, nämlich Browser A. Damit ihre Browser fast jede Webseite korrekt anzeigen konnten, entwickelten die Ingenieure immer kompliziertere und noch dazu undokumentierte Heuristiken, um die Absichten schlampiger Webmaster zu erraten. Dabei wurde oft die Sicherheit und gelegentlich auch die Kompatibilität geopfert. Leider förderte jede dieser Änderungen die Praktiken schlechten Webdesigns[7] noch mehr und zwang die restlichen Hersteller dazu, bei diesem Unsinn mitzumachen, um überleben zu können. Das Fehlen eines ausführlichen, aktuellen Standards hat mit Sicherheit nicht dazu beigetragen, die Ausbreitung dieser Krankheit zu bremsen.

Um die Auswirkungen der galoppierenden Entwicklungsanarchie zu mildern und um die Erweiterung von HTML zu steuern, gründeten Tim Berners-Lee und eine Handvoll Unternehmen als Sponsoren im Jahre 1994 das World Wide Web Consortium (W3C). Leider konnte diese Organisation jedoch lange Zeit nur hilflos zusehen, wie das Format wahllos erweitert und verdreht wurde. Die ersten Arbeiten des W3C an HTML 2.0 und 3.2 versuchten lediglich, auf den Stand des Status quo zu kommen. Das führte zu halbgaren Spezifikationen, die bei ihrer Veröffentlichung schon größtenteils veraltet waren. Das Konsortium versuchte

6. Beispielsweise wollte Microsoft nicht mit Sun verhandeln, um die Lizenz für die Marke JavaScript zu erwerben (eine Sprache, die nur aus werbetechnischen Gründen so genannt wurde, obwohl sie nichts mit Java zu tun hat). Also nannte das Unternehmen seine fast gleiche, aber nicht identische Version »JScript«. In der offiziellen Dokumentation von Microsoft wird die Sprache nach wie vor so bezeichnet.
7. Paradebeispiele für törichte und letzten Endes fatale Browserfunktionen sind die Mechanismen zum Erkennen von Inhalten und Zeichensätzen, die beide in Kapitel 13 besprochen werden.

auch, an einigen neuen und gut durchdachten Projekten zu arbeiten, z.B. Cascading Style Sheets (CSS), hatte es aber schwer, die Unterstützung der Hersteller zu gewinnen.

Andere Bemühungen, bereits implementierte Mechanismen zu standardisieren oder zu verbessern, vor allem HTTP und JavaScript, gingen von anderen Stellen aus, z.B. der ECMA (European Computer Manufacturers Association), der ISO (International Organization for Standardization) und der IETF (Internet Engineering Task Force). Leider waren diese Bemühungen meistens nicht miteinander koordiniert, und manche Diskussionen und Designentscheidungen wurden von Herstellern oder anderen Beteiligten beherrscht, die sich nicht um die langfristigen Aussichten der Technologie kümmerten. Ergebnisse waren eine Reihe toter Standards, widersprüchliche Ratschläge und einige erschreckende Beispiele gefährlicher Interaktionen zwischen eigentlich sauber entworfenen Protokollen. Letzteres Problem wird besonders deutlich werden, wenn wir in Kapitel 9 eine Reihe von Isolationsmechanismen besprechen.

1.2.3 Das Zeitalter der Langeweile: 2000 bis 2003

Als der Streit um das Web abflaute, wuchs die Vorherrschaft von Microsoft infolge seiner Geschäftsstrategie, den Browser mit dem Betriebssystem zu bündeln. Zu Beginn des nächsten Jahrzehnts war Netscape Navigator auf dem Weg in die Bedeutungslosigkeit, während der Internet Explorer einen beeindruckenden Marktanteil von 80% hielt – ein Wert in der Größenordnung, die Netscape noch fünf Jahre zuvor beanspruchen konnte. Auf beiden Seiten gehörten die Sicherheit und die Interoperabilität zu den bedeutendsten Opfern des Krieges um Features und Funktionen. Es bestand jedoch die Hoffnung, nun wo die Kämpfe vorbei waren, dass die Entwickler ihre Differenzen beilegen und zusammenarbeiten konnten, um das Durcheinander gerade zu biegen.

Stattdessen führte die Vorherrschaft jedoch zur Selbstgefälligkeit: Nachdem Microsoft seine Ziele in großem Maße erreicht hatte, gab es für das Unternehmen kaum Anreize, umfangreich in den Browser zu investieren. Bis Version 5 erschien jährlich ein neues Release des Internet Explorer (IE). Doch bis Version 6 aufkam, dauerte es zwei Jahre, und bis zur Aktualisierung von IE6 auf IE7 sogar fünf. Angesichts des Desinteresses von Microsoft hatten andere Hersteller nur wenig Ansatzpunkte, um weitreichende Änderungen durchzuführen, denn die Betreiber der meisten Websites waren nicht gewillt, Verbesserungen vorzunehmen, die nur bei einem kleinen Bruchteil ihrer Besucher funktionierten.

Hinweis

Der Internet Explorer beherrschte bereits in Version 5.5 viele Features, die heute unter dem Namen HTML5 zusammengefasst werden – nur in wunderlicher und nichtstandardisierter Art und Weise. Auch das Samenkorn für Web 2.0 und die JavaScript-Renaissance wurden durch Features im Internet Explorer 5.5 mitbestimmt. So ist beispielsweise das nachfolgend erwähnte XMLHttpRequest-Objekt in Gestalt eines ActiveXObject erstmals im IE implementiert worden. Ähnliches gilt für Vektorgrafiken, Animationen, Video-Elemente, Iframes mit Sandbox und viele weitere Details.

Als positiv erwies es sich jedoch, dass die Verlangsamung der Browserentwicklung dem W3C die Gelegenheit gab, aufzuholen und einige Konzepte für die Zukunft des Web sorgfältig zu untersuchen. Zu den neuen Initiativen, die um 2000 abgeschlossen waren, gehörten HTML 4 (eine bereinigte Sprache, in der viele der redundanten oder unerwünschten Merkmale aus früheren Versionen entfernt oder als veraltet gekennzeichnet waren) und XHTML 1.1 (ein striktes und wohlstrukturiertes Format auf XML-Basis, das leichter und eindeutiger zu analysieren ist und keine proprietären Heuristiken zulässt). Außerdem nahm das Konsortium erhebliche Verbesserungen am DOM (Document Object Model) von JavaScript und an CSS vor. Am Ende des Jahrhunderts war das Web jedoch schon zu stark gewachsen, um einige der alten Sünden auszumerzen. Andererseits war es noch so jung, dass die Sicherheitsprobleme nicht drängend und deutlich genug erschienen. Die Syntax wurde verbessert, Tags wurden ausgemustert, Validatoren wurden geschrieben und kosmetische Korrekturen wurden vorgenommen. Die Browser aber blieben größtenteils, wie sie waren: aufgebläht, eigensinnig und unvorhersehbar.

Aber schon bald geschah etwas Bemerkenswertes: Microsoft schenkte der Welt eine anscheinend unbedeutende, proprietäre API mit dem verwirrenden Namen `XMLHttpRequest`. Dieser triviale Mechanismus sollte eigentlich nichts Besonderes sein, sondern nur eine kleine Lücke in der Webversion von Microsoft Outlook beseitigen. Es stellte sich aber heraus, dass `XMLHttpRequest` zu weit mehr zu gebrauchen war und eine nahezu unbeschränkte asynchrone HTTP-Kommunikation zwischen clientseitigem JavaScript und dem Server erlaubte, ohne dass dadurch Seiten zeitaufwändig und störend neugeladen werden mussten. Damit trug die API zum Aufkommen dessen bei, was später als *Web 2.0* bezeichnet wurde – eine Reihe komplexer, ungewöhnlich schnell reagierender browsergestützter Anwendungen, in denen die Benutzer an vielschichtigen Datenmengen arbeiten, gemeinsam Inhalte erstellen und veröffentlichen konnten usw. Dies griff im weiteren Verlauf auch auf das unantastbare Gebiet der »echten«, installierbaren Clientsoftware über. Verständlicherweise sorgte das für einen ziemlichen Aufruhr.

1.2.4 Web 2.0 und die zweiten Browserkriege: 2004 und später

In Verbindung mit der großen Beliebtheit des Internets und der breiten Verfügbarkeit von Browsern verschob `XMLHttpRequest` die Grenzen des Web hin zu neuen Ufern – und bescherte uns eine Menge Sicherheitslücken, die Privatanwender und gewerbliche Nutzer gleichermaßen trafen. Um das Jahr 2002 herum waren Würmer und Sicherheitsschwächen im Browser ein häufiges Thema in den Medien. Durch die Marktmacht von Microsoft und den damit verbundenen trägen Umgang mit Sicherheitslücken stand das Unternehmen besonders im Fokus. Die Firma hatte das Problem zwar heruntergespielt, doch es entwickelte sich eine Atmosphäre, die einer kleiner Rebellion glich.

2004 tauchte ein neuer Herausforderer unter den Browsern auf: Mozilla Firefox. Dabei handelte es sich um einen von der Community unterstützten Nachfolger des Netscape Navigator, der gezielt die Sicherheitsmängel und die fehlende Standardisierung des Internet Explorer anging. Von Journalisten und IT-Experten gepriesen, erreichte Firefox schnell einen Marktanteil von 20%. Obwohl sich bald herausstellte, dass der Neuling eine fast genauso lückenhafte Sicherheit aufwies wie sein Gegenstück aus Redmond, halfen ihm Open Source und die Unabhängigkeit von Unternehmen dabei, Sicherheitslücken wesentlich schneller zu schließen.

Hinweis

Warum sollten die Anbieter so fieberhaft miteinander konkurrieren? Im Browsermarkt verdient man mit seinen Marktanteilen ja kein Geld. Daher haben Experten den Wettstreit lange für eine Frage der Macht gehalten: Denn durch Bundling, Promotions oder das Anbieten bestimmter Onlinedienste (und seien sie so einfach wie die voreingestellte Suchmaschine) würde der Hersteller des Browsers einen großen Teil des Internets kontrollieren.

Auch ohne Firefox hatte Microsoft Sorgen. Sein Flaggschiff, das Windows-Betriebssystem, wurde immer mehr zu einer (entbehrlichen?) Startplattform für den Browser, und mehr und mehr Anwendungen verlagerten sich ins Web (von Textverarbeitung bis hin zu Spielen). Das war keine gewünschte Entwicklung.

Diese Entwicklung – in Verbindung mit der plötzlichen Popularität von Apples Safari und vielleicht auch den Fortschritten von Opera bei den mobilen Geräten – musste bei der Microsoft-Geschäftsleitung Besorgnis auslösen. Man hatte das Potenzial des Internets in den frühen 1990er-Jahren unterschätzt, und diesen Fehler wollte man nicht wiederholen. So gab man Dampf bei der Entwicklung des Internet Explorer und schickte die deutlich verbesserten und teils auch sichereren Versionen 7, 8 und 9 in schneller Folge ins Rennen.

Die Mitbewerber konterten mit neuen Funktionen und dem Anspruch, noch standardkonformer zu sein (wenn auch nur oberflächlich) sowie sichereres und

schnelleres Surfen zu bieten. Überrascht durch den Erfolg von `XMLHttpRequest` und ohne die Lehren aus der Vergangenheit zu berücksichtigen, entschlossen sich die Anbieter, massiv mit neuen Ideen zu experimentieren, wobei manchmal halbgarer Code oder unsichere Entwürfe herauskamen, wie *globalStorage* bei Firefox oder *httponly*-Cookies beim Internet Explorer.

Um das Bild noch weiter zu komplizieren, bildete einige vom W3C frustrierte Mitarbeiter ein völlig neues Standardisierungsgremium namens Web Hypertext Application Technology Working Group (WHATWG). WHATWG ist nach wie vor an der Entwicklung von HTML5 beteiligt, einer erstmalig ganzheitlichen und auch auf Sicherheit ausgerichteten Revision bestehender HTML-Standards. Berichten zufolge wird diese von Microsoft jedoch unter anderem aus patentrechtlichen Gründen gemieden. Die Entwickler des Internet Explorer verlassen sich auf den eher statistischen HTML5-Snapshot, der vom W3C propagiert wird.

Durch seine ganze Geschichte hindurch hat das Web eine einzigartige Entwicklung durchgemacht, mit starkem, schnellem Wettbewerb, oft höchst politisch und manchmal ziellos, ohne eine einigende Vision und ohne eine einheitliche Sicht auf das Thema Sicherheit. Dieser Mangel an Orientierung hat Browser in ihrer Arbeitsweise und in ihrem Umgang mit Benutzerdaten bis heute nachhaltig negativ beeinflusst.

Vermutlich wird sich die Lage in absehbarer Zeit nicht ändern.

1.3 Die Entwicklung einer Bedrohung

Browser und die dazugehörigen Dokumentformate und Kommunikationsprotokolle haben sich offensichtlich auf ungewöhnliche Weise entwickelt. Das mag zwar die große Zahl von Sicherheitslücken erklären, aber das allein deutet noch nicht darauf hin, dass diese Probleme einzigartig oder nennenswert sind. Um dieses Kapitel abzurunden, werfen wir daher noch einen kurzen Blick auf die speziellen Eigenschaften der häufigsten Onlinebedrohungen und zeigen, warum sie in den Jahren vor dem Web keine Entsprechungen hatten.

1.3.1 Der Benutzer als Sicherheitslücke

Die vielleicht wichtigste (und ganz und gar nicht technische) Eigenschaft von Browsern ist, dass die meisten Personen, die sie benutzen, kaum tief greifende Kenntnisse besitzen. Seit Anbeginn der Computerära waren ungeschulte Anwender immer ein amüsantes Randproblem. Die Beliebtheit des Web – in Kombination mit einer geringen Eintrittshürde – bedeutet aber, dass wir einem neuen Feind gegenüberstehen: der großen Mehrheit von Anwendern, die einfach nicht genug wissen, um sich sicher zu verhalten.

Lange Zeit, so scheint es, hatten die Entwickler von allgemeiner Software nur sehr willkürliche Vorstellungen von dem Vorwissen ihrer Anwender. Die meisten

ihrer Annahmen hatten auch keine ernsten Konsequenzen. Die falsche Bedienung eines Texteditors hat z.B. nur geringen oder keinen Einfluss auf die Systemsicherheit, denn inkompetente Anwender bekommen lediglich ihre Arbeit nicht erledigt – darüber hinaus passiert nichts Schlimmes.

So funktionieren Browser aber nicht. Anders als komplizierte Spezialsoftware können Browser von Personen eingesetzt werden, die fast gar keine Computerkenntnisse besitzen, von Personen, die womöglich nicht mal einen Editor bedienen können. Gleichzeitig können Browser jedoch nur von solchen Personen *sicher* eingesetzt werden, die ein umfassendes Verständnis der IT und der entsprechenden Fachsprache besitzen, z.B. was die Public-Key-Verschlüsselung angeht. Es ist offensichtlich, dass diese Kenntnisse den Anwendern der heute am meisten genutzten Webanwendungen fehlen.

Browser haben noch immer eine Oberfläche, als ob sie von Computerfreaks für Computerfreaks erstellt wären, inklusive kryptischer oder inkonsistenter Fehlermeldungen, diverser komplizierter Konfigurationseinstellungen und einer verwirrenden Vielzahl von Sicherheitswarnungen und -meldungen. Eine beachtenswerte Studie von Wissenschaftlern aus Berkeley und Harvard ergab 2006, dass sorglose Anwender beinahe alle Signale ignorierten, die einen Entwickler alarmiert hätten, etwa das Vorhandensein oder Fehlen von Schloss-Symbolen in der Statusleiste [4]. In einer anderen Studie kamen Stanford- und Microsoft-Wissenschaftler zum selben Ergebnis, als sie den Einfluss des modernen Sicherheitsindikators »grüne URL-Zeile« untersuchten. Dieser Mechanismus, der eine intuitivere Darstellung als das Schloss ermöglichen sollte, machte es tatsächlich einfacher, einem Anwender vorzugaukeln, dass ein bestimmter Grün-Ton Sicherheit bedeutete, unabhängig davon, wo sich die Farbe befand [5].

Experten argumentieren, dass die fehlende Einsicht eines durchschnittlichen Anwenders nicht die Schuld der Softwarehersteller sei und daher kein Entwicklungsproblem darstelle. Andere halten es für unverantwortlich, dass eine leicht erhältliche und weit verbreitete Software Anwender dazu zwingt, ständig sicherheitskritische Entscheidungen zu treffen, die nicht für die eigentliche Nutzung der Software erforderlich sind.

Nur die Browserentwickler zu beschuldigen ist jedoch ebenso unfair: Niemand in der IT-Industrie hat Lösungsansätze für diesen Bereich und es gibt kaum Studien über die Entwicklung von Benutzerschnittstellen (UI), die verhältnismäßig komplex und gleichzeitig sicher sind. Wir schaffen das ja nicht einmal für Geldautomaten.

1.3.2 Die Cloud oder die Freuden gemeinschaftlichen Lebens

Eine weitere, eigenartige Eigenschaft des Web ist die kaum erkennbare Trennung zwischen verschiedenen unabhängigen Anwendungen und den von ihnen verarbeiteten Daten.

Im traditionellen Modell, das von beinahe allen PCs der vergangenen 15 Jahre verfolgt wurde, gab es klare Grenzen zwischen Datenobjekten (Dokumenten), Benutzerprogrammen (Anwendungen) und dem Betriebssystemkern. Letzterer regelt die Kommunikation zwischen den Anwendungen und der Hardwareein- und -ausgabe (I/O) und setzt Sicherheitsregeln für den Fall durch, dass sich eine Anwendung eigenartig verhält. Diese Grenzen sind wohlbekannt und hilfreich, um praktische Sicherheitsmechanismen umzusetzen. Beispielsweise wird eine in Ihrem Editor geöffnete Datei kaum Ihre E-Mail ausspionieren, wenn nicht ein wirklich unglückliches Zusammenspiel von Implementierungsmängeln all diese Schichten auf einmal durchbricht.

In der Welt der Browser gibt es diese Trennung praktisch nicht: Dokument und Code existieren jeweils als Teile ein und desselben, durchmischten HTML-Dokuments. Eine Isolierung zwischen voneinander unabhängigen Anwendungen findet bestenfalls partiell statt (wobei sich alle nominell eine globale JavaScript-Umgebung teilen), und viele Arten von siteübergreifender Interaktion werden implizit erlaubt.

In gewissem Sinne erinnert dieses Modell an CP/M, DOS und andere Single-Task-Betriebssysteme ohne robusten Speicherschutz, präemptives CPU-Scheduling oder Multiuser-Merkmale. Der offensichtliche Unterschied ist, dass diese frühen Betriebssysteme kaum dazu genutzt wurden, um gleichzeitig mehrere, nicht vertrauenswürdige und von Angreifern manipulierbare Anwendungen ablaufen zu lassen. Deshalb musste man sich darüber auch keine Sorgen machen.

Letzten Endes ist das scheinbar unwahrscheinliche Szenario, dass ein Textdokument Ihre E-Mail ausspioniert, im Web aber frustrierend häufig. So ziemlich alle Webanwendungen müssen sich daher nach Kräften um die Abwehr bösartiger domänenübergreifender Zugriffe kümmern und mühsam für eine zumindest grundlegende Trennung von Code und angezeigten Daten sorgen. Aber früher oder später versagen fast alle Webanwendungen. Denn Schwachstellen, die auf der Manipulation von Inhalten basieren – etwa Cross-Site-Scripting oder Cross-Site Request Forgery – sind extrem verbreitet und treffen nur selten auf sichere Client-Architekturen.

1.3.3 Unvereinbare Vorstellungen

Glücklicherweise ist die Situation in der Browserlandschaft in puncto Sicherheit nicht vollständig hoffnungslos. Trotz der nur geringen Zahl von Trennmechanismen zwischen Webanwendungen bieten verschiedene selektive Sicherheitstechniken einen rudimentären Schutz gegen die offensichtlichsten Angriffe. Dies führt uns jedoch zu einem weiteren Merkmal, das das Web zu einer so interessanten Materie macht: Es gibt kein gemeinsames, ganzheitliches Sicherheitsmodell, an das man sich halten kann. Wir sind nicht auf der Suche nach Weltfrieden, wie Sie sich erinnern, sondern nach einer einfachen Kombination flexibler Paradigmen,

die auf die meisten, wenn schon nicht auf alle Fälle des relevanten Sicherheitsmodells zutreffen. In der Unix-Welt ist beispielsweise das *rwx*-Benutzer/Gruppen-Zugriffsmodell ein solcher einheitlicher Ansatz. Aber im Reich der Webbrowser?

Im Reich der Webbrowser könnte man einen Mechanismus namens *Same-Origin Policy* als Kandidaten für einen solchen Sicherheitsansatz bezeichnen. Das gilt jedoch nur, bis man erkennt, dass dadurch lediglich eine sehr kleine Menge von Interaktionen zwischen Domänen geregelt wird. Abgesehen davon gibt es nicht weniger als sieben verschiedene Spielarten, wobei jede davon Sicherheitsgrenzen zwischen Anwendungen zieht – jeweils mit minimalen Abweichungen.[8] Mehrere Dutzend weitere Sicherheitsmechanismen, unabhängig von der Same-Origin Policy, steuern andere Schlüsselaspekte des Browserverhaltens (und implementieren dabei im Wesentlichen, was der jeweilige Autor seinerzeit für den besten Ansatz eines Sicherheitsmodells hielt).

So gibt es Hunderte kleiner, cleverer Hacks, die in ihrer Gesamtheit nicht unbedingt ein funktionierendes Sicherheitssystem ergeben. Das Fehlen einer Integration macht es schon schwierig zu entscheiden, wo eine einzelne Anwendung endet und eine andere beginnt. Wie stellt man vor diesem Hintergrund fest, wo Angriffe stattfinden können? Wie vergibt oder entzieht man Zugriffsrechte oder erfüllt andere sicherheitsrelevante Aufgaben? Oft genug ist »Daumen drücken« der bestmögliche Ratschlag, den wir geben können.

Seltsamerweise machen gutgemeinte Ansätze, die Sicherheit durch weitere Kontrollen zu verbessern, das Problem noch schlimmer. Viele dieser Ansätze legen neue Sicherheitsgrenzen fest, die nicht perfekt mit den vorhandenen Regeln harmonieren, sondern etwas daneben liegen. Wenn die neuen Mechanismen feingranularer sind, werden sie in Verbindung mit alten Regeln oftmals unwirksam und wiegen den Anwender in falscher Sicherheit. Wenn sie gröber sind, könnten sie Zusicherungen außer Kraft setzen, die im Web zurzeit gültig sind. (Adam Barth und Collin Jackson betrachten das Thema destruktiver Beeinflussung verschiedener Sicherheitsansätze von Browsern in ihrer Arbeit [6][7].)

1.3.4 Browserübergreifende Interaktion: Synergie des Versagens

Die insgesamte Störanfälligkeit eines Ökosystems, das aus verschiedenen Softwareprodukten besteht, könnte einfach als Summe der Einzelgefährdungen der Anwendungen betrachtet werden. In einigen Fällen mag die resultierende Gefährdung auch geringer sein (die Diversität verbessert ja die Widerstandsfähigkeit), auf keinen Fall sollte sie jedoch größer sein – würde man meinen.

---

8. Die sieben Hauptvarianten, die in Teil II dieses Buches beschrieben werden, umfassen die Sicherheitsrichtlinien für den JavaScript-Zugriff auf das DOM, die *XMLHttpRequest*-API, HTTP-Cookies, lokale Speicher-APIs und Plug-ins wie Flash, Silverlight und Java.

Hier ist das Web wieder die Ausnahme von der Regel. Die Sicherheits-Community hat eine beträchtliche Anzahl von Fällen entdeckt, die keinem bestimmten Quelltextstück zugeordnet werden können, jedoch eine wirkliche Bedrohung darstellen, wenn verschiedene Browser miteinander interagieren. Kein einzelnes Produkt ist hier als Übeltäter zu erkennen. Alle erledigen ihre Aufgabe, und das einzige Problem besteht darin, dass niemand sich um gemeinsame Verhaltensregeln gekümmert hat.

Beispielsweise mag ein Browser gemäß seinem Sicherheitsmodell davon ausgehen, dass er bestimmte URLs an andere Anwendungen geben darf oder dass er bestimmte Daten auf Festplatte speichern bzw. von dort abrufen darf. Für jede solche Annahme gibt es vermutlich mindestens einen Browser, der damit nicht einverstanden ist und davon ausgeht, dass andere Parteien stattdessen seine Regeln befolgen. Die Gefahr, so etwas für Angriffe auszunutzen, wird dadurch verschärft, dass Browseranbieter versuchen, eine Webseite einfach so auf ihren eigenen Browser umzuleiten, ohne dass der Anwender sein Einverständnis dazu gibt. So erlaubt Firefox das Öffnen von Seiten durch das Protokoll *firefoxurl:*, Microsoft installiert sein eigenes .NET-Gateway-Plug-in in Firefox, und Chrome tut dasselbe mit dem Internet Explorer mittels eines Protokolls namens *cf:*.

Hinweis

Besonders bei solchen Interaktionen ist es unsinnig, einer bestimmten Partei die Schuld zuzuweisen. In einem vor einiger Zeit bekannt gewordenen Fall bei *firefoxurl:* beschuldigten Microsoft und die halbe Sicherheits-Community Mozilla, während Mozilla und die andere Hälfte der Experten Microsoft die Schuld zuschoben [8]. Es spielt keine Rolle, wer Recht hatte: Das Resultat war ein Riesenschlamassel.

Weitere, eng verwandte Probleme (die in den Tagen vor dem Web praktisch unbekannt waren) sind die Inkompatibilitäten der Sicherheitsmechanismen, die in den einzelnen Browsern implementiert sind und sich nur vordergründig gleichen. Wenn sich die Sicherheitsmodelle aber unterscheiden, dann kann ein bestimmter Entwicklungsansatz bei der einen Webanwendung sinnvoll sein, bei einer anderen Anwendung aber unpassend und fehlgeleitet. Tatsächlich können verschiedene Arten rudimentärer Aufgaben (z.B. das Ausliefern einer von einem Benutzer erstellten Textdatei) nicht in allen Browsern sicher implementiert werden. Diese Tatsache ist Entwicklern jedoch erst dann klar, wenn sie mit einem der betroffenen Browsern arbeiten – und auch dann müssen sie erst mit einer entsprechenden Situation konfrontiert werden.

Schließlich tragen alle in diesem Abschnitt beschriebenen Mechanismen zu einer neuen Klasse von Sicherheitslücken bei, die in der Taxonomie vielleicht aufgenommen werden als *Versagen, undokumentierte Vielfalt zu berücksichtigen*. Viele Exemplare tummeln sich in dieser Klasse.

1.3.5 Die Aufhebung der Grenze zwischen Client und Server

Forscher auf dem Gebiet der Informationssicherheit erfreuen sich an einer Welt statischer, klar zugewiesener Rollen, die einen vertrauten Referenzpunkt bei der Zuordnung von Sicherheitsinteraktionen in einer ansonsten komplizierten Welt darstellen. Zum Beispiel sprechen wir von Alice und Bob, zwei hart arbeitenden Anwendern, die miteinander kommunizieren möchten, und Mallory, dem gewieften Angreifer, der sie ausspähen möchte. Dann gibt es die Clientsoftware (im Wesentlichen unintelligente, manchmal einfache I/O-Terminals, die leichtfertig Dienste anfordern) und Server, die die Launen des Clients demütig befolgen. Entwickler kennen diese Rollen und spielen mit, indem sie ziemlich nachvollziehbare und testfähige Umgebungen im Netzwerk entwerfen.

Das Web begann als klassisches Beispiel einer reinen Client-Server-Architektur, aber die funktionalen Grenzen zwischen Client und Server verschwammen schnell. Der Übeltäter war JavaScript, eine Sprache, die es HTTP-Servern ermöglichte, Anwendungslogik in den Browser (den Client) zu verlagern, und ihnen zwei überzeugende Gründe lieferte, dies auch zu tun. Erstens führt diese Verlagerung zu besser reagierenden Benutzerschnittstellen, da Server nicht jede erdenkliche Zustandsänderung der Schnittstelle mitmachen müssen. Zweitens können CPU- und Speicherausstattung des Servers (und damit dessen Kosten) erheblich reduziert werden, wenn Clientrechner rund um den Erdball an der Arbeit beteiligt werden.

Der Diffusionsprozess der Client-Server-Architektur begann recht unschuldig, aber es war nur eine Frage der Zeit, bis auch Sicherheitsmechanismen auf die Clientseite ausgelagert wurden, zusammen mit all der anderen alltäglichen Funktionalität. Warum sollte z.B. HTML auf der Serverseite aufwändig zurechtgebastelt werden, wenn die Daten auch per JavaScript beim Client zusammengefügt werden konnten?

In einigen Fällen wurde dieser Trend extrem ausgenutzt, so dass aus dem Server schließlich ein schlichtes Speichergerät wurde, während beinahe alle Konvertierungs-, Bearbeitungs-, Anzeige- und Konfigurationsaufgaben im Browser landeten. In diesen Szenarien konnte die Serverabhängigkeit sogar umgangen werden, indem Offline-Erweiterungen, wie die persistente Speicherung von HTML5, genutzt wurden.

Eine Verlagerung der Anwendungslogik ist nicht zwingend eine große Sache, allerdings gilt das nicht gleichermaßen für die Verschiebung aller Sicherheitszuständigkeiten an den Client. Selbst in den Fällen, in denen der Server als einfaches Speichergerät fungiert, dürfen die Clients nicht beliebig Zugriff auf die Daten aller Anwender erhalten, und man sollte sie auch nicht damit betreuen, die Zugriffsrechte selbst zu verwalten. Da es nicht wünschenswert war, die Sicherheitslogik der Anwendung beim Server zu belassen, und weil es unmöglich war, sie komplett auf den Client zu übertragen, landeten die Anwendungen schließlich

irgendwo zwischen Client und Server. Eine klar erkennbare und logische Aufgabenteilung zwischen Client- und Serverkomponenten gab es nicht mehr, denn die dabei entstehenden ungewöhnlichen Strukturen und Verhaltensmuster der Anwendungen hatten einfach kein Gegenstück in der Rollenverteilung, die bis dahin in der Security-Welt galt.

Dies führte nicht nur zu einem Durcheinander auf Designebene, sondern auch zu einer Komplexität, die man nicht mehr los wird. In einem traditionellen Client-Server-Modell mit wohldefinierten APIs kann man das Serververhalten prüfen, ohne den Client einbeziehen zu müssen und umgekehrt. Zusätzlich ist es in jeder dieser Komponenten möglich, einzelne Funktionsblöcke zu isolieren und Schlüsse zu ziehen, welche Aufgabe sie erfüllen. Im neuen Modell, zu dem noch undurchsichtige APIs im Web hinzukommen, sind diese Analysewerkzeuge schlicht hinfällig, und damit ist auch eine einfache Bewertung der Sicherheit eines Systems nicht mehr möglich.

Das unerwartete Versagen standardisierter Sicherheitsmodelle und Testprotokolle ist ein weiteres Problem, das das Web zu einem besonderen – und besonders unheimlichen – Platz im Universum der Informationssicherheit macht.