Code-injection Verwundbarkeit in Web Anwendungen am Beispiel von Cross-site Scripting

Martin Johns

Universität Passau
mj@sec.uni-passau.de

Abstract: Cross-site Scripting (XSS) dominiert seit einigen Jahren die Verwundbarkeitsstatistiken. Die hier zusammengefasste Dissertation präsentiert eine methodische Behandlung dieser Klasse von Sicherheitsproblemen. Wir bearbeiten das Problem in drei Schritten: (1) Nach einer Darlegung der notwendigen technischen Grundlagen erfassen wir über eine Klassifikation der Menge der existierenden Angriffstypen das Bedrohungspotential von XSS. (2) Darauf aufbauend entwickeln wir geeignete technische Schutzmethoden vor XSS Angriffen zur Laufzeit. (3) Schlussendlich, zeigen wir, wie XSS Verwundbarkeiten durch das Erzwingen von strikter Daten/Code Trennung auf Sprachebene grundsätzlich und verlässlich verhindert werden können.

1 Einleitung

Web Anwendungen sind heutzutage allgegenwärtig: Ihre Verwendung spannt sich vom Online-Banking über Textverarbeitung und Email bis zu Nutzerinterfaces von Hardware Geräten, wie Netzwerkroutern oder digitalen Bilderrahmen.

Parallel zum Anstieg der Bedeutung von Web Anwendungen hat auch die Menge an dokumentierten Verwundbarkeiten im Zusammenhang mit diesem Anwendungsparadigma zugenommen. Derartige Verwundbarkeiten, wie etwa Cross-site Scripting, SQL Injektion, PHP File Inclusion oder Directory Traversal, dominieren aktuell die entsprechenden Statistiken [CM07] und ein Ende dieses Trends ist nicht abzusehen (siehe Abb. 1). Innerhalb dieser Verwundbarkeitstypen ist Cross-site Scripting am stärksten vertreten.

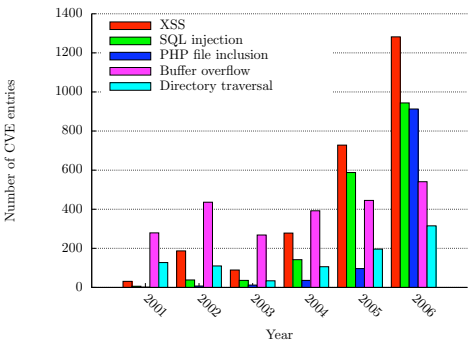


Abbildung 1: Verwundbarkeitsstatistik [CM07]

Die in diesem Artikel zusammengefasste Dissertation ist eine methodische Bearbeitung des Themas Cross-site Scripting (XSS). Die Arbeit gliedert sich in drei Hauptteile: Eine systematischen Erfassung des Verwundbarkeitsschemas und der sich daraus ergebe-

nen Angriffsarten (siehe Abschnitt 2), der Entwurf von Methoden zur dynamischen Abwehr von XSS-basierten Angriffen (siehe Abschnitt 3) und der Erarbeitung der grundlegenden Ursachen der Verwundbarkeitsklasse zusammen mit einer Entwicklung von programmiersprach-basierten Techniken, die das Problem ursächlich angehen (siehe Abschnitt 4). Die folgenden Abschnitte dieses Artikels folgen der Struktur der Dissertation.

2 Cross-site Scripting (XSS) in Web Anwendungen

Mit dem Begriff XSS werden alle Sicherheitsprobleme zusammengefasst, die es dem Angreifer erlauben im Ausführungskontext einer verwundbaren Web Anwendung beliebigen JavaScript-Code im Web Browser des Angriffsopfers auszuführen. Für ein genaues Verständnis von XSS ist es notwendig, klar zwischen der zugrundeliegenden *XSS Verwundbarkeit* und dem resultierenden Angriff (von nun an als *XSS Payload* bezeichnet) zu unterscheiden:

XSS Verwundbarkeiten sind die Sicherheitslücken, welche dem Angreifer das Einschleusen des Schadcodes erlaubt. Basierend auf den grundlegenden Spezifikationen und dokumentierten Techniken, legt die Dissertation die Ursachen dar, die eine Web Anwendung anfällig für XSS Schwächen machen können. In diesem Zusammenhang werden sowohl verschiedenen Klassen von unsicherer Programmierung als auch Szenarios, in denen fehlerhafte Infrastruktur zu XSS Verwundbarkeiten führen können, behandelt.

XSS Payloads: Dokumentierte Angriffe haben gezeigt, das XSS Payloads in ihren maliziösen Möglichkeiten nicht auf Angriffsziele innerhalb der verwundbaren Anwendung (wie z.B. dem Session Cookie) beschränkt sind [GHPR07]. Vielmehr können solche Attacken auch andere Ziele, wie beispielsweise die lokale Ausführungsumgebung des Nutzers oder Rechner in dessen Intranet, haben [Joh08]. Um eine methodische Erarbeitung der potentiell möglichen Angriffsklassen zu ermöglichen, werden zuerst die grundlegende Eigenschaften von sowohl JavaScript wie auch von HTML und HTTP dargelegt. Darauf aufbauend präsentiert die Dissertation ein vier-stufiges, hierarchisches Klassifikationsschema, welches das systematische Erfassen von XSS Payloads erlaubt (siehe Abb. 2). Basierend auf dieser Klassifikation präsentiert die Arbeit erstmals einen umfassenden, strukturierten Überblick der bis dato dokumentierten XSS Payload Typen.

Anders als verwandte Sicherheitsprobleme, wie z.B. SQL Injection, weist XSS ein dreigliedriges Angriffsszenario auf, das neben dem Angreifer und dem Opfer noch die anfällige Web Anwendung als Mittler beinhaltet (siehe Abb. 3). Daraus folgt, dass der Verantwortliche für die XSS Verwundbarkeit (der Web Server Betreiber) und das Opfer des Angriffs (der Nutzer bei dem die XSS Payload aus-

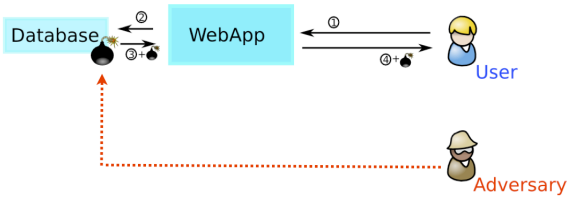


Abbildung 3: XSS Angriff

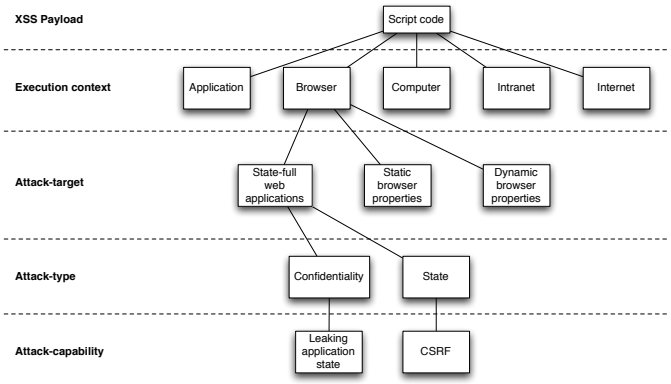


Abbildung 2: Auszug aus dem Klassifikationsschema für XSS Payloads

geführt wird) sind zwei unterschiedliche Entitäten mit nur loser Kopplung. Diese Eigenschaft hat zwei Konsequenzen: Zum einen sind aktive XSS Angriffe serverseitig sehr schwer zu erkennen, da die eigentliche Ausführung des Schadcodes im Browser des Opfers stattfindet [JEP08]. Ausserdem hat das eigentliche Opfer keine effektive Möglichkeit sich ursächlich gegen XSS Payloads zu wehren[1], da sich der Source Code der verwundbaren Anwendung ausserhalb seiner Zugriffsmöglichkeiten befindet.

Aus diesen Beobachtungen ergeben sich zwei komplementäre defensive Strategien:

1. *Entwicklung von Methoden zur aktiven Abwehr von potentiellen Angriffen zur Laufzeit*. Das Ziel solcher Methoden ist es, den Nutzern von Web Anwendungen Schutzmöglichkeiten zu geben, die effektiv sind für den Fall, dass die verwendete Anwendung eine XSS Schwäche aufweist. Wir beschreiben unsere Resultate in dieser Kategorie im Abschnitt 3.

2. *Grundlegende Vermeidung von Verwundbarkeiten auf der Sprachebene*. Die große Mehrheit aller XSS Schwächen wird durch unsicheres Programmieren verursacht. Folglich ist ein Ansiedeln einer ursächlichen Lösung auf der Source Code Ebene zielführend. Wir beschreiben unsere Resultate in diesem Bereich in Abschnitt 4.

3 Angriffsabwehr zur Laufzeit

Wie oben motiviert, stehen dem eigentlichen Angriffszielen von XSS Attacken, dem Nutzer der verwundbaren Web Anwendung, keine effektiven Schutzmethoden zur Verfügung. Aus diesem Grund beschäftigt sich die Dissertation mit dem Entwurf von defensive Methoden, die kein Beseitigen der unterliegenden XSS Verwundbarkeit benötigen. Stattdes-

[1]Dem Nutzer bleibt die Möglichkeit JavaScript vollständig zu dekativieren. Dieses ist aber bei modernen Web 2.0 Anwendungen (wie z.B. Google Docs) keine realistische Option.

sen adressieren die entwickelten Techniken direkt die Ausführung des Schadcodes durch das Einführen von geeigneten Einschränkungen in der Laufzeitumgebung der Anwendung, welche ausgewählte Angriffstypen verhindern ohne die eigentliche Funktionsweise der Anwendung zu beschränken. Derartige Techniken sind zwangsläufig immer speziell auf einzelne Klassen von Angriffstypen (sprich XSS Payloads) ausgerichtet. Die Dissertation präsentiert eine Methodik zum Entwickeln solcher Schutzmassnahmen und stellt drei von uns entworfene und implementierte Techniken vor.

3.1 Methodik

Die im Rahmen der Dissertation entwickelten Schutzmassnahmen wurden nach folgender Methodik entworfen:

(a) Zuerst wird eine fein-granulare Analyse der betrachteten Klasse von XSS Payloads (z.B. „Session Hijacking") erstellt. Spezieller Fokus dieser Analyse ist die Identifizierung der einzelnen technischen Schritte, die ein Angreifer durchlaufen muss, um sein Ziel zu erreichen. Wichtig in diesem Zusammenhang ist es, alle Angriffsvarianten zu berücksichtigen, die in der betrachteten Payload Klasse existieren (in Falle von „Session Hijacking" sind diese „SID theft", „Browser hijacking" und „Background XSS propagation" [Joh06]). Andernfalls besteht die Gefahr, dass die entwickelte Gegenmassnahme nur unvollständigen Schutz bietet.

(b) Basierend auf dieser Analyse, wird eine minimale Menge an technischen Fähigkeiten isoliert, die der Angreifer zwingend besitzen muss, um sein Vorhaben durchzuführen (z.B. Lesezugriff auf Cookie-daten via JavaScript im Falle von „SID Theft").

(c) Im Anschluss, werden geeignete Verfahren untersucht, die dem Angreifer bzw. dem vom Angreifer kontrollierten JavaScript zur Laufzeit die identifizierten Fähigkeiten entziehen, ohne das reguläre Funktionieren der Anwendung zu behindern.

Im Zusammenhang mit dieser Methodik ist die Klassifikation von XSS Payloads aus Abschnitt 2 von hoher Relevanz, da diese einen systematischen Blick auf das bestehende Angriffspotential erlaubt. Dieses erlaubt in Schritt (a) die Identifizierung von verwandten XSS Payload Typen. Im Anschluss an die eigentliche Entwicklung der Schutzmassnahme in Schritt (c) kann die Klassifizierung verwendet werden, um die erzielte Schutzabdeckung zu ermitteln. Dieses Vorgehen kam im Rahmen der Dissertation bei der Evaluation der vorgeschlagenen Verfahren zum Einsatz.

3.2 Entwickelte Techniken

Im Rahmen der Dissertation wurden Gegenmassnahmen für die XSS Payload Klassen „Session Hijacking", „Cross-site Request Forgery" und „Angriffe auf Intranet Ressourcen" entwickelt. In diesem Abschnitt stellen wir diese kurz vor. Da für eine genaue Dar-

stellung der jeweiligen Funktionsweisen in diesem Artikel der Platz fehlt, verweisen wir für weitere Details auf die Dissertation oder die zugehörigen Publikationen.

Session Hijacking [Joh06]: Unter dem Begriff „Session Hijacking" werden XSS Payloads zusammengefasst, die es dem Angreifer erlauben, auf die verwundbare Anwendung unter der Identität des Opfers zuzugreifen. Basierend auf der oben vorgestellten Methodik, werden drei komplementäre Techniken vorgestellt: „Deferred Loading", „One-time URLs" und „Subdomain Switching". Jede dieser Techniken entwaffnet jeweils eine der identifizierten Angriffsvarianten (siehe oben).

Cross-site Request Forgery [JW06]: Um den Autorisierungskontext eingehender HTTP Requests zu ermitteln, finden in der Kommunikation zwischen Browser und Server verschiedene Techniken Verwendung. Bei einer Reihe von diesen Techniken fügt, nach der initialen User Authentifizierung, der Browser die Autorisierungsinformation automatisch zu den HTTP Requests hinzu. Dieses findet transparent für den User und die Anwendung statt. Dieses Verhalten ermöglicht dem Angreifer im Browser des Users per JavaScript versteckte HTTP Requests an fremde Anwendungen zu schicken, die von dem Server als autorisiert erkannt werden. Auf diese Weise kann der Angreifer gegebenenfalls Zustandsverändernde Aktionen im Namen des Nutzers auslösen. XSS Payloads die dieses Angriffsschema implementieren, sind unter dem Namen „Cross-site Request Forgery" bekannt. Im Rahmen der Dissertation wurde ein Client-seitiger Proxy entwickelt, der dem Angreifer die Möglichkeit entzieht, cross-site HTTP Requests zu erzeugen, die valide Autorisierungsinformationen beinhalten.

Angriffe auf Intranet Ressourcen [JW07]: Um das Intranet vom Internet zu trennen, finden Firewalls Verwendung, die allen ungewollten eingehenden Netzwerkverkehr blocken. Allerdings wird ein Web browser im Allgemeinen *innerhalb* des Intranets ausgeführt. Durch die technischen Möglichkeiten von JavaScript (im eingeschränkten Umfang) HTTP Requests zu erzeugen, ist es dem Angreifer möglich, vom Browser des Nutzers aus Intranet Ressourcen anzugreifen [Joh08]. Zur Abwehr von entsprechenden XSS Payloads wurde eine Web Browsererweiterung konzipiert und implementiert, die dem Angreifer die Möglichkeit nimmt, über ein extern-bezogenes JavaScript, HTTP Requests an Ziele innerhalb des Intranets zu schicken.

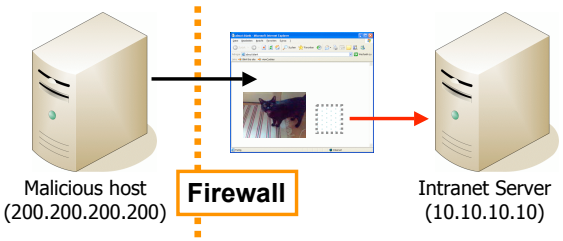


Abbildung 4: Angriff auf Intranet Ressourcen

4 Grundlegende Vermeidung von XSS Verwundbarkeiten auf der Sprachebene über Daten/Code Trennung

4.1 String-based Code Injection

Genauer betrachtet sind die Verwundbarkeitsklassen Cross-site Scripting, SQL Injection, Remote Command Injection und Path Traversal allesamt Varianten des selben Verwundbarkeitsmusters (siehe Listing 1) - der *String-based Code Injection*: Innerhalb einer Programmiersprache (der *nativen Sprache*, z.B. PHP) werden über den String Datentyp dynamisch Syntax-Fragmente einer anderen Computersprache (die *Fremdsyntax*, z.B. HTML oder SQL) zusammengesetzt und im Anschluss an einen anwendungsexternen Parser (z.B. der Web Browser oder die Datenbank) geschickt, um dort interpretiert zu werden. Wenn der Programmierer in diesem Prozess dynamische Daten verwendet, die vollständig vom Angreifer kontrolliert werden können, kann dieses dazu führen, dass der Angreifer die Semantik der so generierten Fremdsyntax verändern kann.

```
1 // Dynamische Daten aus dem HTTP Request
2 $data = $_GET("data");
3 // XSS Verwundbarkeit
4 print "<a href='http://www.foo.org'>"+ $data+ "</a>";
5 // SQL Injection Verwundbarkeit
6 mysql_query("SELECT * FROM users where passwd =" + $data);
```

Listing 1: Beispiele für verwundbare Fremdsyntax Generierung

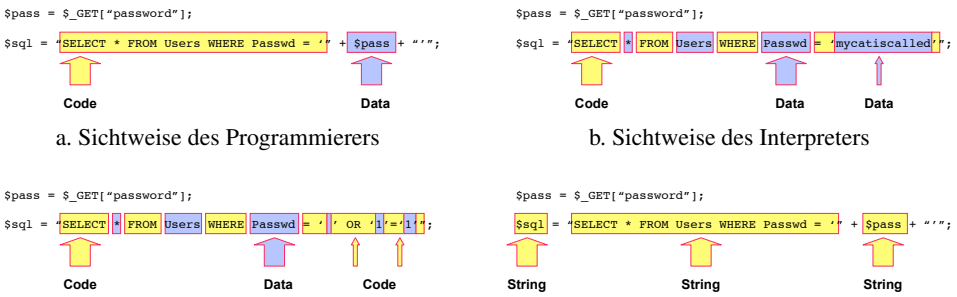
Um sich gegen solche Unsicherheiten zu schützen, muss der Programmierer bei allen nicht-statischen Bestandteilen der Fremdsyntax Generierung geeignete Codierungstechniken anwenden, um eventuelle Injektionsangriffe zu verhindern.

4.2 Grundlagen der Daten/Code Trennung

Die Evidenz der vergangenen Jahre [CM07] zeigt, dass die oben beschriebene, manuelle Technik zur sicheren Programmierung nicht dazu beiträgt, die Anzahl der entstehenden Verwundbarkeiten substantiell zu verringern. Daraus resultiert das Bedürfnis nach einer ursächlichen Lösung, welche dieses Muster der unsicheren Programmierung schlicht nicht mehr zulässt.

Das verursachende Problem der beschriebenen Sicherheitsprobleme ist das implizite Vermischen von Daten und Code bei der Fremdsyntax Generierung. Nehmen wir das Beispiel[2] in Abbildung 5: Die intuitiven Annahmen des Programmierers bzgl. Daten und Code Anteile in der generierten Fremdsyntax (Abb. 5.a) entsprechen nicht der technischen, grammatik-getriebenen Interpretation des Parsers (Abb. 5.b). Dieses kann von dem Angreifer ausgenutzt werden, um zusätzliche Code Fragmente einzuschleusen (Abb. 5.c). Der verwendete String Datentyp der nativen Sprache seinerseits stellt dem Programmierer keinerlei Möglichkeiten bereit, seine ursprünglichen Annahmen bzgl. Daten und Code zu

[2] Im folgenden wird in den Beispielen auf die Sprache SQL zurückgegriffen. Diese Wahl liegt in der vergleichsweise knappen syntaktischen Struktur von SQL begründet, welche kurze Beispiele erlaubt.



a. Sichtweise des Programmierers

b. Sichtweise des Interpreters

c. Sichtweise des Interpreters (Code Injection)

d. Sichtweise der Sprache

Abbildung 5: Unterschiedliche Sichtweisen auf dynamische Codegenerierung

kommunizieren (Abb. 5.d).

Als Konsequenz aus dieser Beobachtung folgt, dass Verfahren gefunden werden müssen, die eine strikte Trennung zwischen Daten und Code bei der dynamischen Generierung von Fremdsyntax erzwingen.

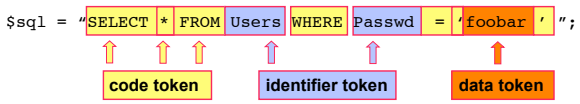


Abbildung 6: Identifizierte Token Klassen

Vor dem Entwurf solcher Techniken muss allerdings zuvor geklärt werden, wie genau die Konzepte *Daten* und *Code* auf eine Computersprach-Syntax anzuwenden sind. Zu diesem Zweck präsentiert die Dissertation ein methodisches Vorgehen, welches eine systematische Klassifizierung der Token-typen erlaubt, die in der formalen Grammatik der betrachteten Sprache definiert sind. Diese Token werden in die Klassen *Code Token*, *Identifier Token* und *Data Token* (siehe Abb. 6) aufgeteilt.

In den folgenden Abschnitten präsentieren wir zwei Methoden, die String-based Code Injection Verwundbarkeiten bekämpfen, indem sie verhindern, dass ein Angreifer Code Token in dynamisch erzeugte Syntax einzuschleusen kann. Für genauere Details und Ergebnisse der jeweiligen praktischen Evaluierungen verweisen wir auf die Dissertation bzw. die korrespondierenden Publikationen.

4.3 Dynamische Trennung von Daten und Code durch String Masken [JB07]

Eine Untersuchung des Quell-Codes von bestehenden Web Anwendungen führt zu folgender Erkenntnis: Vor dem Ausführen des Programms ist die syntaktische Struktur der im Programm enthaltenen Fremdsyntax bereits vollständig definiert. Das heisst, alle Code Token der Fremdsyntax sind bereits als vordefinierte Teile von statischen Stringwerten vorhanden. Syntaxfragmente die zur Laufzeit hinzugefügt werden, dienen lediglich dem Zweck die generierte Syntax mit Datenwerten (also Data Token) zu ergänzen. Basierend

auf dieser Prämisse, stellt das entwickelte Verfahren eine Methode vor, alle Code Token, die als statische Elemente im Quell-Code vorhanden sind, innerhalb der Stringwerte der Anwendung eindeutig zu markieren. Dieses passiert über das Anwenden von randomisierten Stringmasken, welche innerhalb der String-Literale die Code Token temporär ersetzen. Bevor die Fremdsyntax an die anwendungsfremden Interpreter gegeben wird, kann diese auf unmaskierte Code Token untersucht werden. Da alle legitim existierenden Code Token zuvor mit den Stringmasken überschrieben wurden, sind alle gefundenen Code Token zur Laufzeit über dynamische Datenwerte in die Strings gekommen und somit höchst wahrscheinlich Teil einer String-based Code Injection Attacke. Solche Code Token werden unschädlich gemacht, danach werden die Stringmasken wieder durch die legitimen Code Token ersetzt und die Fremdsyntax kann sicher an die externen Interpreter weitergereicht werden. Das eigentliche Anwenden und Entfernen der Stringmasken passiert dabei über zentral positionierte Pre- und Post-Prozessoren (siehe Abb. 7).

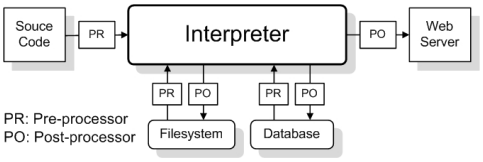


Abbildung 7: Pre- und Postprozessor

4.4 Einführung eines dedizierten Datentyps zur Fremdsyntax-Erzeugung [JBGP10]

Wie schon in Abschnitt 4.2 erläutert, bietet der native String Datentyp dem Programmierer keinerlei Möglichkeiten zur Trennung von Daten und Code bei der dynamischen Generierung von Fremdsyntax. Aus diesem Grund präsentiert die Dissertation eine programmiersprachbasierte Lösung, die das Verfahren der Fremdsyntax Erzeugung durch den String Typ mit geeigneteren Methoden ersetzt. Kern der vorgestellten Lösung ist eine Erweiterung des Typsystems der nativen Programmiersprache um einen Datentypen, der das konstruieren von Fremdsyntax erlaubt und dabei eine mandatorische, strikte Trennung zwischen Daten und Code erzwingt. Zur Integration dieses Datentyps in die bestehende Web Anwendungsarchitektur wurden ausserdem zwei weitere Kernkomponenten konzipiert: Eine Sprachintegration, die dem Programmierer ein einfaches Definieren der Fremdsyntax erlaubt und eine Abstraktionsschicht, die den neu eingeführten Datentyp sicher in eine charakter-basierte Fremdsyntax übersetzt, welche dann an die externen Interpreter kommuniziert werden kann (siehe Abb. 8).

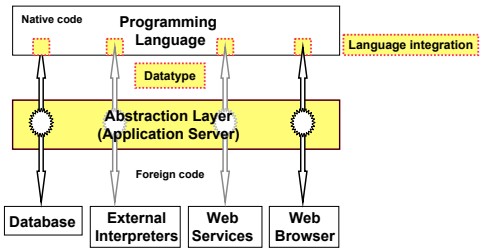


Abbildung 8: Schlüsselkomponenten

Um eine formale Validation des Konzeptes zu erlangen, wurde die vorgeschlagene Erweiterung des Typsystems der nativen Programmiersprache typ-theoretisch modelliert (siehe Abb. 9). Dabei wurden analog zu [VSI96] neue Integritätstypen eingeführt, welche die

$$CT \subseteq IT \subseteq DT$$

$$\text{(base)}\ \ \Gamma \vdash e : \tau$$

$$\text{(r-val)}\ \frac{\Gamma \vdash e : \tau\ var}{\Gamma \vdash e : \tau}$$

$$\text{(cmd}^{-}\text{)}\ \frac{\tau \subseteq \tau'}{\tau'\ cmd \subseteq \tau\ cmd}$$

$$\text{(subtype)}\ \frac{\Gamma \vdash e : \tau \qquad \tau \subseteq \tau'}{\Gamma \vdash e : \tau'}$$

$$\text{(trans)}\ \frac{\tau \subseteq \tau' \qquad \tau' \subseteq \tau''}{\tau \subseteq \tau''}$$

$$\text{(assignment)}\ \frac{\Gamma \vdash v : \tau\ var \qquad \Gamma \vdash e : \tau}{\Gamma \vdash v := e : \tau\ cmd}$$

Abbildung 9: Typ-theoretischer Kalkül (Auszug)

Konzepte *Daten* und *Code* abbilden. Auf diese Weise kann über eine Variante des klassischen Biba-Modells [Bib77] nachgewiesen werden, dass es in dem vorgestellten System keine Informationsflüsse von Daten zu Code geben kann und somit zuverlässig vor Code Injection geschützt wird. Zum Nachweis der praktischen Verwendbarkeit des Systems zum Schutz vor XSS Verwundbarkeiten diente eine Implementierung der beschriebenen Kernkomponenten für den J2EE Applikationsserver. Neben verschiedenen funktionalen Tests wurde diese Implementierung verwendet, um eine nicht-trivial, große (JSPWiki, 69.712 Lines of Code) auf unser System zu portieren. Die ausgewählte Version der Software wies ursprünglich diverse XSS Schwächen auf. Nach der Portierung zu unserem System waren keine der zuvor bestehenden Verwundbarkeiten mehr vorhanden.

5 Zusammenfassung

Die Ergebnisse der Dissertation ermöglichen eine ganzheitliche Behandlung der Problems XSS durch das Eröffnen einer unmittelbaren und einer langfristigen Handlungsoption:

Kurz- und mittelfristig werden wir mit einer Situation umgehen müssen, in der XSS Schwächen Teil der Anwendungslandschaft sind. Solange dieses der Fall ist, sind die entwickelten Verfahren zur Angriffsabwehr (siehe Abschnitt 3) von hoher Relevanz. Weiterhin haben wir mit unserem Angriffsklassifikationsverfahren und unserer Methodik zur systematischen Entwicklung von Schutzmethoden zwei Werkzuge eingeführt, die dazu beitragen neuen Angriffsklassen zeitnah und effektiv begegnen zu können.

Langfristig ist es notwendig das Problem der String-based Code Injection grundsätzlich anzugehen. Wir haben mit unserem Ansatz zur sprachbasierten Daten/Code Trennung bei der Generierung von Fremdsyntax (Abschnitt 4) eine Methodik vorgestellt, welche dieses Ziel zuverlässig und praktikabel erreichbar macht. Eine breite Annahme unseres Ansatzes in der praktischen Software Entwicklung würde dazu führen, dass die Hauptursache der Verwundbarkeitsklasse effektiv ausgeschaltet wird.

Literatur

[Bib77] Kenneth J. Biba. Integrity Considerations for Secure Computer Systems. Bericht MTR-3153, Mitre Corporation, April 1977.

[CM07] Steve Christey und Robert A. Martin. Vulnerability Type Distributions in CVE, Version 1.1. [online], `http://cwe.mitre.org/documents/vuln-trends/index.html`, (09/11/07), May 2007.

[GHPR07] Jeremiah Grossman, Robert Hansen, Petko Petkov und Anton Rager. *Cross Site Scripting Attacks: XSS Exploits and Defense*. Syngress, 2007.

[JB07] Martin Johns und Christian Beyerlein. SMask: Preventing Injection Attacks in Web Applications by Approximating Automatic Data/Code Separation. In *22nd ACM Symposium on Applied Computing (SAC 2007), Security Track*, Seiten 284 – 291. ACM, March 2007.

[JBGP10] Martin Johns, Christian Beyerlein, Rosemaria Giesecke und Joachim Posegga. Secure Code Generation for Web Applications. In *2nd International Symposium on Engineering Secure Software and Systems (ESSoS '10)*, Jgg. 5965 of *LNCS*, Seiten 96 – 113. Springer, February 2010.

[JEP08] Martin Johns, Bjoern Engelmann und Joachim Posegga. XSSDS: Server-side Detection of Cross-site Scripting Attacks. In *Annual Computer Security Applications Conference (ACSAC'08)*, Seiten 335 – 344. IEEE Computer Society, December 2008.

[Joh06] Martin Johns. SessionSafe: Implementing XSS Immune Session Handling. In *European Symposium on Research in Computer Security (ESORICS 2006)*, Jgg. 4189 of *LNCS*, Seiten 444–460. Springer, September 2006.

[Joh08] Martin Johns. On JavaScript Malware and Related Threats - Web Page Based Attacks Revisited. *Journal in Computer Virology, Springer Paris*, 4(3):161–178, August 2008.

[JW06] Martin Johns und Justus Winter. RequestRodeo: Client Side Protection against Session Riding. In *Proceedings of the OWASP Europe 2006 Conference, refereed papers track, Report CW448*, Seiten 5 – 17., Katholieke Universiteit Leuven, May 2006.

[JW07] Martin Johns und Justus Winter. Protecting the Intranet Against "JavaScript Malware" and Related Attacks. In *Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2007)*, Jgg. 4579 of *LNCS*, Seiten 40 – 59. Springer, July 2007.

[VSI96] Dennis M. Volpano, Geoffrey Smith und Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4:167–187, 1996.

Martin Johns studierte Mathematik und Informatik in Göttingen, Santa Cruz (CA) und Hamburg, wo er 2003 sein Diplom in Informatik erhielt. Von 1994 bis 2005 arbeitete er ausserdem als Software Ingenieur in verschiedenen deutschen Unternehmen (u.A. Infoseek Germany, TC Trustcenter und SAP AG). 2005 trat er dem Lehrstuhl von Joachim Posegga bei (zunächst in Hamburg, später dann in Passau), um in verschiedenen Forschungsprojekten in den Bereichen Software- und Web Applikations-Sicherheit zu wirken. Seit Ende 2009 bekleidet er die Position eines Senior Reseachers für Security and Trust bei SAP Research am Standort Karlsruhe. Seine aktuellen Arbeitsthemen umfassen den Schutz von kritischen Infrastrukturen, Future Energy und Anwendungssicherheit.