Quantifying the Attack Surface of a Web Application

Thomas Heumann, Jörg Keller
University of Hagen
Dept. of Mathematics and Computer Science
58084 Hagen, Germany
joerg.keller@fernuni-hagen.de

Sven Türpe
Fraunhofer SIT
Rheinstraße 75
64295 Darmstadt, Germany
sven.tuerpe@sit.fraunhofer.de

Abstract: The attack surface of a system represents the exposure of application objects to attackers and is affected primarily by architecture and design decisions. Given otherwise consistent conditions, reducing the attack surface of a system or an application is expected to reduce its overall vulnerability. So far, only systems have been considered but not single applications. As web applications provide a large set of applications built upon a common set of concepts and technologies, we choose them as an example, and provide qualitative and quantitative indicators. We propose a multi-dimensional metric for the attack surface of web applications, and discuss the rationale behind. Our metric is easy to use. It comprises both a scalar numeric indicator for easy comparison and a more detailed vector representation for deeper analysis. The metric can be used to guide security testing and development. We validate the applicability and suitability of the metric with popular web applications, of which knowledge about their vulnerability already exists.

1 Introduction

Measuring security properties is challenging yet necessary. The need to make informed decisions implies necessity, while the complex and sometimes counter-intuitive nature of security makes measuring it a challenge. Various security metrics for various purposes have been proposed, but meaningful quantification of security remains an open problem [Ver09].

We contribute to this ongoing endeavour a metric for the attack surface of web applications, capturing a set of exposure factors in the architecture and design of such applications. Our attack surface metric differs from existing approaches in two important aspects: the metric is tailored to a class of applications, and it does not require access to the application's source code. For a more in-depth discussion see [Heu10]. Security practitioners often use the term *attack surface* colloquially to refer to the amount of code, functionality, and interfaces of a system exposed to attackers [How04]. The idea behind this notion is that attackers can exploit vulnerabilities only in those parts of a system that the system exposes to adversaries. Comparing two otherwise equal systems, exposed to equal populations of adversaries, the one with the smaller attack surface is statistically expected to be less vulnerable. Since other factors—e.g. code quality and defect distribution, security mechanisms, or user behaviour—independently influence the security of a system, this expected correlation may not always be observed in real-world settings where such factors cannot be controlled.

So far, there have been few attempts to formalize the intuitive notion of an attack surface. A notable exception is the seminal work by Manadhata [Man08], Manadhata and Wing [MW05, MW10] and Howard, Pincus and Wing [HPW05]. They consider the attack surface from a system's perspective and establish a generic formal notion based on the dimensions *targets and enablers*, *channels and protocols*, and *access rights*. Their apparent starting point are threats in conjunction with common rules of thumb [Pei08], such as the principle of least privilege and the recommendation to disable unnecessary network services. Consequently, their attack surface definition and metric is generic though perhaps slightly tailored to network hosts with network services. They provide a formal justification and underpinning for measures reducing the attack surface of a system.

Web applications more and more become the standard type of application software. They are built upon a common set of concepts and technologies: browsers as generic clients; HTTP as the communication protocol between browsers and web servers; server-side and client-side runtime environments for code execution; and XML as a universal technology for data representation. Their common platform brings about common security considerations and common security problems of web applications, as illustrated by the OWASP Top Ten Most Critical Web Application Security Risks [OWA10]. Securing a web application is therefore different from securing a web server host, and it is a task primarily for application developers rather than system administrators. To this end the developer has to accomplish a variety of tasks: properly employ security mechanisms and functions, avoid certain classes of software defects, and follow design principles and best practices, and reduce the attack surface of the application, to name just a few. An attack surface metric helps with the latter task. It makes effects measurable and applications or versions comparable. We present a metric to measure the attack surface of a web application and discuss its rationale. We validate its applicability with known web applications.

The remainder of this paper is organized as follows. Section 2 briefly introduces web applications. In Sect. 3 we describe our approach to quantifying the attack surface of such applications. The parameters used to estimate attack surfaces are described in Sect. 4. Section 5 demonstrates how the metric is applied in an experimental evaluation. We conclude in Sect. 6.

2 Web Applications and Their Attack Surface

Web Applications A typical web application comprises three subsystems: the browser, the server, and back-end systems (see Fig. 1). The *browser* constitutes the platform for the user interface. It renders HTML code, images, and other data visible, handles user input, and provides the runtime environment for client-side code. Helper applications, such as a PDF reader, and plug-in components, e.g. for Java or Flash, may extend the capabilities of the browser. A browser process at a point in time, and often also the entire browser installation, is associated with a particular user as a personal software tool. The *web server*

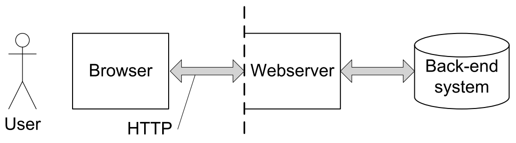

Figure 1: High-level architecture of a typical web application

waits for browsers to connect and answers their HTTP requests. Each request comprises a URL pointing to some resource and possibly additional parameters. The server responds either by serving a static resource, or by executing a program and returning the output of this program to the requesting browser. In the course of fulfilling a request the server, or a program executed by the server, may access *back-end systems* such as databases. Contrary to the impression that Fig. 1 might give, multiple instances may exist for each of the subsystems, with complicated relationships between the instances. In particular, a browser may access multiple web servers simultaneously.

Attack Surface of a Web Application The purpose of our attack surface metric is to estimate the amount of functionality and code that a web application exposes to outside attackers. Web applications comprise a natural security boundary, indicated by the dashed line in Fig. 1: users and attackers alike normally are not granted access to the web server and back-end systems other than through the HTTP interface of the web server. We consider any attack that requires such elevated privileges as a prerequisite an insider threat and beyond the scope of this paper. We further exclude from our consideration attacks that primarily target the users of an application, such as phishing attacks to get their passwords. Accessible to the outside attacker—who may or may not also be a user of the application—are therefore the HTTP interface(s) of the web server(s), any server-side or back-end functionality accessible through the server, and any data displayed and code executed on the browser side, including the browser itself. Finally we exclude vulnerabilities of the underlying client or server (operating) system.

A simplistic definition of the attack surface would consider only the HTTP interface of the web server and the server-side functionality accessible through this interface. However, many typical risks [OWA10] involve important client-side aspects. In a Cross-Site Scripting (XSS) attack, for instance, the attacker may exploit server-side software defects to inject JavaScript code into the client-side portion of the application, and exploit the capabilities thus gained to copy session cookies that give access to additional server-side functionality. An attack surface metric should therefore also consider the way the application uses the client subsystem.

3 Approach

Requirements and Assumptions An application attack surface metric is a tool for developers and testers. Developers use it, for instance, to assess the impact of design and implementation decisions, whereas security testers may base on the metric estimates of required testing effort or of the expected number of defects. Our metric is designed with practical applicability for testers in mind, which entails a number of further requirements. We are aiming for these properties:

Attack surface approximation. The metric should represent the attack surface of a web application as outlined in Sect. 2. It should yield low numeric values for a plain old web server with only static content, and growing values as one adds application-level exposures such as interactive functionality.

Universal applicability. The attack surface metric should be applicable to any web site or application regardless of its purpose, functionality, or implementation platform.

Grey-box measurability. There should be practical methods to determine the metric for a given application. Approximate measurements should remain possible if all or some of the application source code is unavailable. This is important for testers as grey-box security testing is common.

Support qualitative comparison. Besides an overall numeric score, the metric should also support comparison on a semi-aggregated level, similar to CVSS [MSR07]. This makes it easier for users of the metric to spot qualitative differences that the overall score would obscure.

Relevant measurements. The metric should use measurements and parameters that commonly vary between applications or versions of an application.

In addition to these requirements we make two assumptions. First, we limit our considerations to external attacks, excluding insider threats from our consideration. Second, we focus on attack surface elements with immediate exposure to external attackers. This means we will not attempt to distinguish applications, for instance, by their use of back-end subsystems or interaction with the operating system of the server host.

Metric Construction We construct our metric using a real vector space of weighted parameters. Every possible attack surface is represented by an *attack surface vector* AS. Raw measurements are mapped into this space according to a set of rules and principles. The Euclidean norm provides us with a straightforward definition for the *attack surface indicator* $ASI = \|AS\|$ as an easy-to-compare numeric value. In addition we can create intermediate representations of the attack surface that preserve some qualitative information by defining several subspaces, projecting AS into each of them, and using the norms of the projections.

Raw measurements can be Boolean values representing presence or absence of a feature, enumerations representing multiple-choice measurements, or non-negative integer values as the result of counting. We represent a Boolean measurement by 0 for *false* or 1 for *true*; an n-ary enumeration by the consecutive sequence of natural numbers from 0 through $n-1$; and integer values by themselves. If possible, we define the measurement such that a smaller contribution to the attack surface implies a lower raw value.

Raw measurements are mapped to components of AS by means of functions. Most components are simply the measured parameter multiplied with a weight. For some dimensions of the vector space the mapping is a different function of a single or of multiple raw

Value	0	1	2	3	4	5
Raw count	0	$1-2$	$3-5$	$6-9$	$10-14$	$15-20$
Value		6	7	8	9	10
Raw count		$21-27$	$28-35$	$36-44$	$45-54$	$55-\infty$

Table 1: Mapping of unbounded counts to a fixed range

measurements. We use this method with several groups of measurements as described in Sect. 5. We also use this method to map unbounded counts to the bounded range $[0, 10]$ according to Table 1. This is particularly useful where we cannot formally justify any upper boundary, yet expect the vast majority of measurements to stay below a certain value, such as with the number of different cookies or domains used by an application.

Attack Surface Vector The attack surface AS of a web application defined according to our construction is a vector consisting of the following components: degree of distribution $ddist$, page creation method dyn, security mechanisms $\langle security\rangle$, input vectors $\langle input\rangle$, active content $\langle active\rangle$, cookies $cookie$, user roles $role$ and access rights $rights$:

$$AS = \langle ddist; dyn; \langle security\rangle; \langle input\rangle; \langle active\rangle; cookie; role; rights\rangle\ .$$

Angle brackets indicate groups of components; the overall number of dimensions is 22. With measurements adjusted such that each component of the vector takes its maximum value, the attack surface vector looks like this:

$$AS_{max} = \langle 34; 1; \langle 1; 10; 10\rangle\ ; \langle 1; 1; 1; 4; 1; 8\rangle\ ; \langle 5; 7; 8; 6; 8; 10; 10; 4\rangle\ ; 40; 10; 10\rangle\ .$$

4 Attack Surface Parameters

For the first version of our metric, we gather candidates from sources such as the OWASP Top Ten, cf. [Heu10] for a detailed discussion. Among those, we select according to measurability. Each of the raw measurements required can be made or approximated by observing an application at runtime. Weights are chosen such that the impact of each parameter on the overall score ASI conforms to our estimation of the relative importance of this parameter. Future versions of the metric should use empirically validated weights. Table 2 shows an overview of all components of the attack surface vector, which we explain in detail below.

Degree of Distribution The degree of distribution $ddist$ summarizes how an application spans multiple domains. Cross-domain issues are a common source of vulnerability. Distribution makes such issues more likely as it requires the application to work around the same-origin policy enforced by web browsers to separate resources of different origins. Distribution also increases the number of potential attack vectors. To determine the degree of distribution we count the number of subdomains under a common 2nd– or lower-level

Parameter Family	Short Name	Parameters	Range
Degree of Distribution	$ddist$	Subdomains $sdom_{wa}$	$[0, 10]$
		Domains dom_{wa}	$[0, 10]$
		Foreign Domains dom_{ext}	$[0, 10]$
Dynamic Creation	dyn	Dynamic Creation	$\{0, 1\}$
Security Features	$security$	TLS $crypt$	$\{0, 1\}$
		Partial TLS $crypto\text{-}mix$	$\{0, 10\}$
		Validate $validate$	$\{0, 10\}$
Input Vectors	$input$	URL Parameters $urlparam$	$\{0, 1\}$
		Forms $forms$	$\{0, 1\}$
		Hidden Fields $hidden$	$\{0, 1\}$
		Authentication Methods $auth$	$\{0, 1\}$
		Search $search$	$\{0, 2, 4\}$
		File Upload $files$	$\{0, 8\}$
Active Content	$active$	Client-side Scripting (own) js	$\{0, 5\}$
		Client-side Scripting (foreign) js_{ext}	$\{0, 7\}$
		Server-Side Scripting sss	$\{0, 8\}$
		AJAX $ajax$	$\{0, 6\}$
		Java $java$	$\{0, 8\}$
		RIA (own) $flash$	$\{0, 10\}$
		RIA (foreign) $flash_{ext}$	$\{0, 10\}$
		Feeds (foreign) $feeds$	$\{0, 4\}$
Cookies	$cookie$	Own Cookies c_{wa}	$[0, 10]$
		Foreign Cookies c_{ext}	$[0, 10]$
Access Control	$role$	Role	$\{0, 5, 10\}$
	$rights$	Privileges	$\{0, 5, 10\}$

Table 2: Attack surface parameters

domain used by the application; the number of 2nd-level domains used; and the number of foreign domain names accessed when the application is being used. Then we determine the value of each parameter $sdom$ of own subdomains, dom of own domains, and dom_{ext} of foreign domains and subdomains involved according to Table 1, e.g. if there are five foreign domains involved then $dom_{ext} = 2$.

From these measurements we calculate the distribution subscore $ddist = \frac{1}{2} \cdot sdom + dom + 2 \cdot dom_{ext} - 1$. Subdomains of a common parent are sometimes considered equal under the same-origin policy whereas foreign domains indicate that external sites or applications are invoked, hence the weights. We subtract 1 as at least one domain is necessary and used per se.

Page Creation Method The parameter dyn represents the binary distinction between applications that dynamically create pages on the server side and those that do not. If an application uses server-side technologies such as PHP, ASP or JSP the value is 1 ($true$), otherwise 0. Server-side application code is a source of vulnerability.

Security Mechanisms The $security$ group of components of the attack surface vector represents the use of common security mechanisms. Security mechanisms are different from other factors in that their presence *reduces* the attack surface. We consider two mechanisms, Transport Layer Security (TLS) and input validation. In addition to their presence we evaluate how TLS is being used by the application. The $security$ group thus comprises three parameters based on Boolean measurements. Presence of TLS is indicated by $crypt \in \{0, 1\}$, the value 0 expressing that HTTPS is in use somewhere in the application. If pages exist that mix content accessed over TLS with content accessed over plain HTTP, we set $crypto\text{-}mix = 10$, otherwise to 0. The third parameter $validate \in \{0, 10\}$ takes on the value 10 if there is any indication, e.g. from testing, that server-side input validation is not in place or broken.

Input Vectors The HTTP interface between client and server supports a number of input vectors. The more vectors an application uses, the more complex it is. We consider URL parameters $urlparam$, HTML forms $forms$, hidden form fields $hidden$ and HTTP authentication mechanisms $auth$ as Boolean parameters with weight 1; file uploads $files \in \{0, 8\}$ with a higher weight; and the presence of search functions as a ternary value $search \in \{0, 2, 4\}$. The parameter $search$ is 0 if no site search is present, 2 if a locally-implemented mechanism is there, and 4 if the application uses an Internet search engine such as Google or Bing. Note that generally foreign features and embedded code from external sources outweigh internal counterparts, because they are beyond control. File uploads get a higher weight than other input vectors because they are potentially more powerful in the hands of attackers and their handling within applications is often different from the handling of other inputs.

Active Content Active content increases the attack surface on the client side, requiring the user to have plug-ins or helper applications in place and adding client-side code and processes to the application. We capture the use of client-side active content and related technologies as a group of Boolean components with different weights: $js \in \{0, 5\}$ if the application contains JavaScript code; $js_{ext} \in \{0, 7\}$ if it loads such code from a different site; $sss \in \{0, 8\}$ for server-side scripting; $ajax \in \{0, 6\}$ if the use of AJAX is observed; $java \in \{0, 8\}$ representing the use of Java applets; $flash \in \{0, 10\}$ and $flash_{ext} \in \{0, 10\}$ for the use of Adobe Flash or similar Rich Internet Application (RIA) technology; and $feeds \in \{0, 4\}$ if the application integrates, or supports the integration of RSS feeds from other sites. Note that the use of a sandbox by a technology does not imply a lower weight. Client-side issues may still affect the application; the sandbox may be optional, as, for instance, with signed Java applets; or the client-side runtime environment may be vulnerable. The weights represent a rough assessment of relative risk.

Cookies Cookies have various implications for the attack surface: cookies constitute another input channel into the application; they are often used to implement user authentication and session management; cookies can be used to track users; and cookies leave easy-to-observe traces in the browser. For the attack surface metric we create a compound parameter, $cookie$, from two measurements. We count the maximum number of cookies and the number of foreign cookies from other sites that the application sets during a user session. The number of cookies is transformed into an intermediate score $c \in [0, 10]$, and the number of foreign cookies into $c_{ext} \in [0, 10]$ according to Table 1, e.g. if there are six foreign cookies then $c_{ext} = 3$. From these intermediate values we calculate $cookie = c + 3 \cdot c_{ext}$ as a component of the attack surface vector.

Access Control The parameter $role$ reflects the user status and can assume one of three values: unauthenticated (0), i.e. a user is not logged in (and possibly anonymous[1]), authenticated (5) or root (10), which means a) the user is logged-in and linked to an identity, b) certain access rights are associated with his identity. A user can have the following access rights, represented by the parameter $rights$: none (0), limited (5) or root (10).

5 Experimental Evaluation

We validate our attack surface metric by applying it to several web sites. This demonstrates how to use the metric, whether its application is practical in real-world scenarios, and whether the results are in line with our expectations. We expect our metric to yield higher values for ASI if the target is large and complicated, and similar values for similar applications. We also explore the range of values that we get from real applications and compare it to the theoretical extremes.

Measuring Parameters The preferable way of measuring attack surfaces would be by using automated tools. Since the metric is designed such that most components of the attack surface vector can be derived from client-side observations, implementing such a tool as an extension of either a web browser or a testing platform should be straightforward. However, such a tool does not yet exist. We demonstrate the metric using manual measurements.

Reliable measurement requires that the application is being used to a sufficient extent. Without delving into a discussion of possible coverage criteria, we require that every accessible function in the user interface of the application is touched during the measurement process. The measurement process starts with a clean client without any cached content or stale cookies and in a configuration that does not restrict the application or its client-side components in any way.

For the purposes of this paper we used a web browser, Firefox 3.5.8, in a particular configuration. We enabled all pertinent warnings and user permission requests, particularly those related to TLS and to cookies; permitted all third-party interactions; disabled automated features such as the password manager and input auto completion as well as phishing and malware warnings; and configured the browser to clear all private data on exit. This configuration ensures that we start with a clean environment each time we restart the browser, receive popup notifications for some of the information we are interested in (e.g. use of

[1] The anonymity may be repealed by persistent cookies, even if the user is not logged in.

Site or Application	Attack Surface Vector	ASI
min	$\langle 0; 0; \langle 0; 0; 0 \rangle ; \langle 0; 0; 0; 0; 0; 0 \rangle ; \langle 0; 0; 0; 0; 0; 0; 0; 0 \rangle ; 0; 0; 0 \rangle$	0
gaos.org	$\langle 0; 1; \langle 1; 0; 0 \rangle ; \langle 1; 1; 1; 1; 2; 8 \rangle ; \langle 5; 0; 0; 8; 0; 0; 0; 0 \rangle ; 1; 5; 5 \rangle$	14.63
testlab.sit.fraunhofer.de	$\langle 2.5; 1; \langle 1; 0; 0 \rangle ; \langle 1; 0; 0; 0; 0; 0 \rangle ; \langle 5; 7; 0; 8; 0; 10; 0; 0 \rangle ; 6; 0; 0 \rangle$	16.83
BSCW	$\langle 0.5; 1; \langle 0; 0; 0 \rangle ; \langle 1; 1; 1; 1; 2; 8 \rangle ; \langle 5; 0; 0; 8; 0; 0; 0; 0 \rangle ; 1; 10; 5 \rangle$	16.98
cFolders	$\langle 0.5; 1; \langle 0; 0; 0 \rangle ; \langle 1; 1; 1; 1; 2; 8 \rangle ; \langle 5; 0; 0; 8; 0; 0; 0; 0 \rangle ; 1; 10; 5 \rangle$	16.98
Livelink	$\langle 0.5; 1; \langle 0; 0; 10 \rangle ; \langle 1; 1; 1; 1; 2; 8 \rangle ; \langle 5; 0; 0; 8; 8; 0; 0; 0 \rangle ; 2; 10; 5 \rangle$	21.34
last.fm	$\langle 13; 1; \langle 0; 10; 0 \rangle ; \langle 1; 1; 0; 1; 2; 0 \rangle ; \langle 5; 0; 6; 8; 0; 10; 10; 0 \rangle ; 25; 5; 5 \rangle$	35.74
max	$\langle 34; 1; \langle 1; 10; 10 \rangle ; \langle 1; 1; 1; 4; 1; 8 \rangle ; \langle 5; 7; 8; 6; 8; 10; 10; 4 \rangle ; 40; 10; 10 \rangle$	60.79

Table 3: Attack surfaces of sample web sites and applications

TLS, cookies), and that the browser does not unnecessarily interfere with the application or our measurement process. In addition we use an interactive HTTP(S) proxy, such as WebScarab or Paros, to observe HTTP traffic.

5.1 Sample Web Applications

We chose our set of evaluation samples to cover a range of different web sites and applications, but with subsets of multiple similar applications. *Similar* is defined loosely here, meaning just that two applications serve more or less the same purpose and exhibit comparable core functionality. Our samples include:

- Simple web sites primarily serving pages with information. We use `gaos.org` and `testlab.sit.fraunhofer.de` as samples. One is the site of an open source software advocacy group and built using a wiki engine. The other is the web site of the research group of one of the authors. It serves mostly static pages but uses a limited amount of PHP code on the server side and JavaScript on the client side without providing much interactive functionality.
- Several web-based document management systems: Livelink 9.2, BSCW 4.4.5 and cFolders 4.0.0. All three are multi-user applications allowing their users to store and share files. They have been developed independently by different companies using different programming languages.
- The site `last.fm` as a representative of a typical web 2.0 site. This site offers a social network platform for music lovers, comprising, for instance, music streaming functions, profiles of users' listening habits, user-to-user communication features, and advertising.

5.2 Results

Quantitative Comparison Table 3 depicts the attack surfaces measured for the sample applications. The indicator values in the rightmost column exhibit several interesting properties. First, they cover only about one third of the theoretical range. This does not come unexpected: the lower and the upper end of the range represent extreme cases that

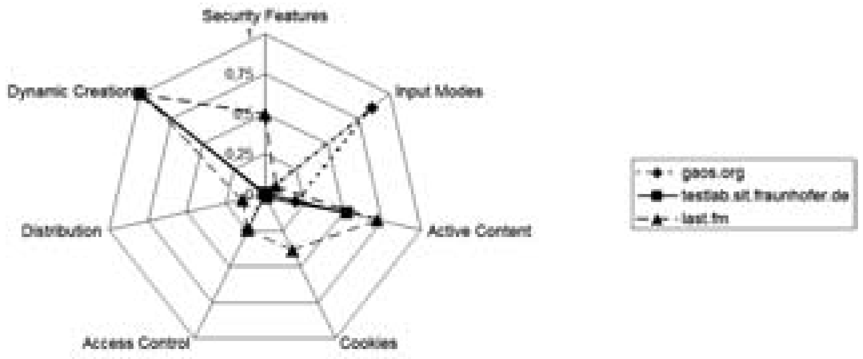

Figure 2: Qualitative comparison of the attack surfaces of web sites

real-world sites or applications are unlikely to reach. Second, and more surprisingly, the two apparently simple web sites reach indicator values close to those of the document management applications considered. This illustrates what the attack surface metric really measures. The two web sites have a high indicator value because they use similar technologies and concepts as web applications. With few exceptions our metric does not consider differences in complexity or size, but rather the mere presence or absence of features. Third, indicator values behave as expected for similar applications. The two simple web sites yield similar values, as do the three document management applications. The higher indicator value of Livelink is largely due to client-side Java applets and known deficiencies in input validation in the (outdated) version considered here. The indicator value clearly separates `last.fm` from the other sites and applications, which seems appropriate considering its nature, purpose, and features.

Analysis of Results Comparing attack surface indicators is a quick way of comparing applications, but the indicator is a lossy compression. Different vectors may lead to similar values. For a detailed analysis and further insight we need to compare the vectors rather than just their norms. One way is to look at the difference of two attack surface vectors, which has non-zero components in just those dimensions in which the two attack surfaces differ. Note that the difference of two attack surface vectors is not always a valid attack surface vector itself.

Due to the number of dimensions the vector difference of two attack surfaces may be difficult to interpret, but we can construct an intermediate view that preserves some of the qualitative information. To this end we calculate intermediate square sums for the seven parameter groups introduced in Sect. 3. Considering their varying ranges imposed by our weighting of parameters, we scale the ranges for each group to the interval $[0, 1]$. Figure 2 shows the resulting radar chart for the three web sites, and Fig. 3 the chart for the three applications. We can immediately see, for instance, that the larger attack surface of `last.fm` is due to the presence of several factors not found in the other samples, whereas for the simpler web sites only some of the factors have an effect at all. We can also see that the attack surfaces of the document management applications are similar.

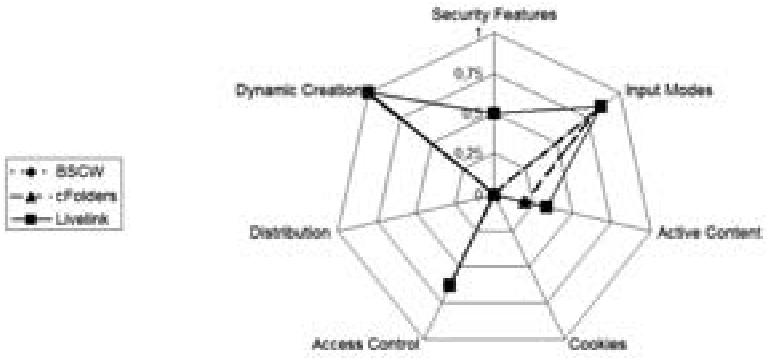

Figure 3: Qualitative comparison of the attack surfaces of document management applications

6 Conclusion

The attack surface metric defined in this paper is a first attempt to measure vulnerability expectations for the class of web applications under grey-box testing conditions. A first evaluation with existing applications indicates a good relation of our proposal with expectations.

In future work, we would like to compare our approach to other metrics, e.g. the traditional source lines of code (SLOC) [Ros97]. Also, we would like to refine our parameters, considering e.g. session management or sensitivity to the type of browser. Finally, it might be worthwhile to investigate thresholds for comparisons of attack surfaces in situations where only a subset of the parameters varies between applications. A common implementation platform, for instance, may fix some of the parameters.

Our attack surface metric is work in progress. The version described in this paper can be improved and refined in various ways:

- Consider further variables. The current version of the metric does not always clearly distinguish simpler web sites from more complex web applications. Considering further features may improve the precision.
- Replace n-ary parameters with counts or percentages where feasible. This, too, may improve the precision of the metric.
- Reconsider weights. We used weights to adjust the impact of individual parameters on the attack surface indicator. If, however, the metric is primarily used for vector comparisons, weights may be less important than we thought.
- Refine the semantics of some of the parameters. Authorization and access control, in particular, are too complex to be properly represented in a few simple parameters.
- Understand the impact of correlations. Not all parameters used in the metric are independent. For instance, each foreign cookie implies involvement of a foreign web site, but not vice versa.
- Properly handle replicated mechanisms. The Flash Player, for instance, has its own scripting language and cookie store. Such alternative mechanisms, when used by an

application, should have just the same impact on the metric as comparable functionality implemented using the primary mechanisms.
- Consider the effects of volatile properties of a web site. The attack surface determined for `last.fm`, for instance, was increased by a banner ad that initiated many connections to foreign sites.

Further work in these directions may ultimately lead to some kind of security metric construction kit providing building blocks and composition rules for the construction of domain– and purpose-specific metrics. The vector approach seems promising as a basis for such work.

Acknowledgement This work was supported by CASED, www.cased.de.

References

[Heu10] Thomas Heumann. Bestimmung der Angriffsfläche von Web-Anwendungen. Master's thesis, Dept. of Mathematics and Computer Science, University of Hagen, 58084 Hagen, `http://publica.fraunhofer.de/documents/N-134399.html`, March 2010.

[How04] Michael Howard. Attack Surface – Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users. `http://msdn.microsoft.com/en-us/magazine/cc163882.aspx`, November 2004.

[HPW05] Michael Howard, Jon Pincus, and Jeannette M. Wing. *Computer Security in the 21st Century*, chapter Measuring Relative Attack Surfaces, pages 109–137. Springer, 2005.

[Man08] Pratyusa Manadhata. *An Attack Surface Metric*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, November 2008. CMU-CS-08-152.

[MSR07] Peter Mell, Karen Scarfone, and Sasha Romanosky. A Complete Guide to the Common Vulnerability Scoring System. `http://www.first.org/cvss/cvss-guide.pdf`, June 2007.

[MW05] Pratyusa Manadhata and Jeannette M. Wing. An Attack Surface Metric. Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, July 2005. CMU-CS-05-155.

[MW10] Pratyusa K. Manadhata and Jeannette M. Wing. An Attack Surface Metric. *Software Engineering, IEEE Transactions on*, Preprint, June 2010.

[OWA10] OWASP Top 10 – 2010. available online, `http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`, 2010.

[Pei08] Holger Peine. Rules of Thumb for Developing Secure Software: Analyzing and Consolidating Two Proposed Sets of Rules. In *ARES '08: Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*, pages 1204–1209, Washington, DC, USA, 2008. IEEE Computer Society.

[Ros97] Jarrett Rosenberg. Some Misconceptions About Lines of Code. In *Fourth International Software Metrics Symposium (METRICS'97)*, volume 0, pages 137–142, Los Alamitos, CA, USA, November 1997. IEEE Computer Society.

[Ver09] Vilhelm Verendel. Quantified security is a weak hypothesis: a critical survey of results and assumptions. In *NSPW '09: Proceedings of the 2009 New security paradigms workshop*, pages 37–50, New York, NY, USA, 2009. ACM.