Information & Security: An International Journal

Mphago et al., vol.40: 2, 2018, 189-202 https://doi.org/10.11610/isij.4014

SELF-ADVERTISING ATTACK SURFACES FOR WEB APPLICATION HONEYPOTS: NEW TECHNOLOGY TO MANAGING CYBER DISASTERS

Banyatsang MPHAGO, Dimane MPOELENG, Shedden MASUPE, and Oteng TABONA

Abstract: Honeypots are security tools often used to attract and learn attackers' methods or divert their attention to unimportant resources. In order to achieve their goal, honeypots need to be identified by the attackers, and this is often the challenge with most deployments of honeypots. This paper therefore explores the use of attack surface sizes in web application honeypots to self-advertise their honeypots to the attackers. PageRank, which is a system that ranks pages on the web through outward link analysis, ranks important pages as seen by their users at the top of the search results. Therefore, the paper argues that vulnerable pages on the web, thus applications with large attack surface, are also important to attackers. Therefore, if pages are ranked based on their importance as seen by their users, pages with large attack surface should rank high when attackers search for them. To design a large attack surface, attack surface parameters can be strategically placed in a template as different parameters affect the attack surface differently when placed in a particular way.

Keywords: honeypot, attack-surface, PageRank, web application, attack surface parameters.

Introduction

Computer Emergency Response Teams (CERTs) are often a group of experts tasked with handling computer security incidents. Once an incident is successfully detected, there is a need to respond. One of the biggest challenges response teams face when responding to incidents is collecting evidence- thus finding out exactly what happened. Getting the evidence of what happened during an attack is not only critical to finding who did the attack but also how the attacks propagated, and thereby readying themselves for future attacks. When a system is compromised, attackers often leave some evidence behind, which when analysed can lead to information on who is the attacker, how he conducted the attack, and what may have been stolen during the attack. Without this information, CERTs are unable to respond correctly to incidents, and this may lead to difficult situations when such attacks occur repeatedly. Howev-

er, analysing the extent of an attack on a live system can be a challenge: evidence can be contaminated or deleted when data on the disks and logs gets re-written by the system's processes. To avoid losing evidence and data getting re-written, systems are often pulled offline for analysis, and this may also result in loss of business. One possible way of curbing this problem is through the use of honeypots. The primary goal of a honeypot is to be attacked and learn the attack methods, or divert attacks to other unimportant resources.

By analysing the evidence trapped in honeypots, no business is lost, but the valuable knowledge on attacks is gained. However, the challenge of using honeypots is that often attackers do not find them. This paper proposes self-advertising attack surfaces that are able to make our honeypots more visible on the Search Engine Optimization (SEO). In the proposed approach, we argue that the size of the attack surface can be used as a factor in marketing the honeypots to its targets. In the next sections of the paper we would give a brief background of honeypot definitions, and attack surfaces in web application honeypots. Then we would present our argument of why the size of attack surfaces in web application honeypots can be used as a form of advertising honeypots to the attackers. Lastly, we would present our approach of maximizing attack surfaces in web application honeypots through the use of attack surface parameters and then conclude by discussing our findings.

Background

A honeypot is tough to define because it is a new and changing technology, and it can be involved in different aspects of security such as prevention, detection, and information gathering.[1] It is unique in that it is more a general technology, not solution, and does not solve a specific security problem. Instead, it is a highly flexible tool with applications in such areas as network forensics and intrusion detection. Some of the definitions of honeypots are listed below:

Definition 1: "a honeypot is a security resource whose value lies in being probed, attacked and compromised."[2]

Definition 2: "a honeypot is a computer which has been configured to some extent to seem normal to an attacker, but actually logs and observes what the attacker does."[3]

Definition 3: "a honeypot is a general computing resource whose sole task is to be probed, attacked, and compromised, used or accessed in any other unauthorised way."[4]

Attack Surfaces in Web Applications

An attack surface of an application is any possible ways in which an attacker can manipulate the application and or possibly cause harm,[5] or an amount of application

area that is exposed for an attack.[6] Thus, attack surface of a system is any system's resource that can be used by an attacker to cause damage to that system or any related elements associated to that system.

Web applications are mostly similar in their concepts: there is a browser that communicates with the webserver through the http protocol, and the webserver will in turn communicate with its backend systems to produce a response back to the browser. As the communication between the browser, webserver, and backend systems takes place, there are several entries and exits points to the web application which either directly or indirectly get used/ exposed to the user. These entry and exit points are what define the attack surface of that web application. The term attack surface is sometimes used literally to refer to the amount of code, functionality, and interfaces of a system exposed to attackers.[5] In web applications, an attack surface can simply be considered as the HTTP boundary (Figure 1) of the server or web application and the functionality running behind the server accessible through that http boundary.

However, client systems, the underlying operating system of the web server, the users, etc., are also part of the attack surface of the web application. But these parameters require some special privileges, i.e. insider information or root privileges before they can be manipulated, and as such would not be discussed in this paper. Rather, this paper would only focus its discussion on what is exposed to the user or attacker through the HTTP interface, assuming that application has the natural security like any normal application.

The Attack Surface Parameters

The degree of distribution of a web application, availability of security features, input vectors, dynamic creation of pages, cookies, active content, and access control methods are the parameters that affect the size of an application's attack surface in either direction.[5, 6] The degree of distribution reflects the traversing of an application across many domains. Cross-domain issues commonly carry vulnerabilities, and distribution enables such problems more possible as it needs the system to abide to the same-origin

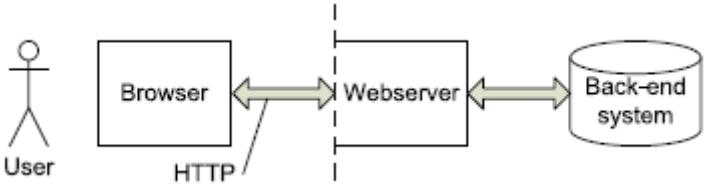

Figure 1: Web Application Architecture.[5]

policy prescribed by browsers to separate sources of multiple origins.[5] Page creation method reflects the difference between pages created dynamically on the server side by the application and those that are static. Application codes that run on the server-side are also a common source of vulnerabilities, so applications that use server-side technologies such as PHP, ASP, or JSP will have a bigger attack surface than the one that is not. The presence of security mechanisms in a webpage reduces the size of the attack surface. These security mechanisms are often the input validation and Transport Layer Security. The http interface between a client and a web server supports a number of input dimensions, and an application with many dimensions will generally be more complex. The more complex an application is, the more vulnerable it becomes, hence a bigger attack surface. Active content on the other hand, affect the attack surface on the client side, where the user uses plug-ins or other side applications in place to add processes and code that do not reside on the server.[5] The availability of cookies in an application has various implications to its attack surface. Cookies bring with them input paths into the application, where these cookies are often used to support session management and user authentication. Cookies can also be used to track users, where they leave observable hints in the browser. All these parameters can an influence the size of the attack surface in either direction. Lastly, access control is a compound parameter that has an influence on the size of the attack surface, and it reflects that a user has role and access rights to certain materials. Access roles defines the user status and can assume one of the three values:[5, 6] unauthenticated, thus the user is not logged in or possible anonymous; authenticated, thus the user is logged and has a known identity; or the user is logged-in and has certain defined access rights. The parameter right reflects that a user can have the following access rights: none, limited, or full rights (i.e. root). All these parameters can either reduce or increase the size of an attack surface of an application.

Attack Surface Size as a Form of Self-Advertising

To learn how to design self-advertising attack surfaces, we must first understand how attackers find vulnerable applications on the web. Basically, attackers use search engines to find vulnerable applications on the web. An example is when an attacker is looking for an instance of a web application with a special kind of vulnerability, and then uses a search engine to perform this special search request. In this case, the search engine would return a list of all potential victims of this particular vulnerability. In this list (search results) of potentially vulnerable applications is where the honeypot should appear if the attackers are to find it. For the honeypot to appear in this search results it should also emulate vulnerable applications that were specified in the search. When attackers find these vulnerable paths (also known as dorks) to the honeypot, they are more likely to attempt to compromise them; and as such one of

the honeypot's goals, which are to be identified by its targets, is realized. When the honeypot gets listed on the search results of vulnerable applications by the search engine, search engine optimization determines where the honeypot is placed within its search results. The most relevant results to the search terms would be placed at the top of the list while the least relevant will be placed at the back of the list. This process is often referred to as Search engine optimization (SEO), which is the process of improving the visibility of a website or a web page in a search engine results.

Assuming that an attacker searched for vulnerable applications on the web and our honeypot got listed in the results, how can one make sure that the honeypot gets listed at the top of the results, or at least closer to the top? Thus, how can the PageRank of the honeypot be as high as possible so that attackers find the honeypot first rather than other vulnerable applications? Our argument is, in addition to the link structure analysis, the attack surface of the honeypot can be used to make the honeypot more visible to the attackers if the size of the attack surface is also factored in the PageRank system, and we call this self-advertising by the honeypot to its targets. A large attack surface of a system does not suggest a system has many weak points to attack but rather such a system is more vulnerable, thus it is easier to attack. A more vulnerable application is therefore attractive to the attackers; hence the honeypot gets the needed attention.

The PageRank algorithm works by analysing the link structure of the web and then measure the authority of the pages based on those links. The goal behind PageRank system is to objectively measure the page's importance as viewed by its users through link structure analysis. Users are interested in important pages (i.e. pages with high authority score). Important pages for attackers however, are not the same as important pages for other conventional web users. Attackers are looking for vulnerable applications on the web; therefore, vulnerable pages are important to attackers. The PageRank system does not rate pages based on vulnerabilities they have, but rather, based on the link analysis structure. In this section we argue that because honeypots are more useful when attacked, in addition to the link structure analysis employed by PageRank system, an attack surface of a honeypot can be used as a factor that influences the PageRank of that honeypot. For web applications, an attack surface can simply be considered as the HTTP boundary of the server or web application and the functionality running behind the server and accessible through that http boundary.

For honeypots, having a large attack surface is desirable. The bigger the attack surface an application has implies the ease at which that application can be exploited. A more exploitable system is a natural attraction to attackers. Therefore, maximizing the attack surface of honeypots suggests an increased importance of that application

to the attackers. Therefore, by combining the importance brought by the size of the attack surface with the link structure analysis employed by the PageRank system, we argue that the PageRank of a vulnerable application would increase if PageRank system uses the size of attack surfaces to rank vulnerable applications. The more vulnerable a honeypot is, the more useful it becomes when attackers want to take advantage of it. By strategically placing attack surface parameters on a web page of a honeypot, the attack surface of that honeypot can be maximized in either direction.

Related Work

A similar approach to our "Self-Advertising Attack Surfaces" proposition is the "Time-Authority Aware Ranking" proposed by Berberich et al.,[7] where they propose a strategy for increasing the PageRank of a page, not based on manipulating out-links but rather based on the recency of information contained in pages. Berberich et al.[7] argue that the freshness of web content and the link structure are factors that should be taken into consideration in link analysis when computing the importance of a page. Therefore, they introduced two link analysis approaches named T-Link and T-Link Light that take into consideration the temporal aspects freshness (i.e. timestamps of most recent updates), and activity (i.e. updates rates) of pages and links.

Another strategy of increasing the PageRank of a page is known as Google bombing, discussed by Hamilton,[8] where Google bombing is described as the activity of designing internet links that will bias the search engine results so as to create an inaccurate impression of the search engine target. In Google bombing, target pages are linked to by many different pages with the same link texts or key phrase, thereby associating the target with the key phrase in Google's PageRank algorithm. Baeza-Yates et al.[9] also studied the impact of collusion, thus nepotistic linking on PageRank manipulation. In this paper Baeza-Yates et al.[9] prove a bound on the PageRank increase that depends both on the reset probability of the random walk ε and on the original PageRank of the colluding set. Thus, a group of nodes can collude to get a higher PageRank by manipulating out-links of the group.

Proposed Framework

Approach

In measuring the attack surface of our application, we selected CVSSv3 (the latest version of CVSS) framework to use as our security metric, mainly because as argued by Markowsky,[10] there is no best metric out there. We therefore regarded the results obtained as estimates of the security level of the application rather than the actual security level. Our goal is to find out by how much the attack surface parameters increase the size of the attack surface of an application when strategically placed,

therefore, the luck of perfect security metric is not a hindrance to achieving our goal. Nessus vulnerability scanner uses CVSSv3 as its security metric in its current version, and as such, Nessus is the tool of choice in estimating the attack surface of our application.

An application's attack surface measurement does not reflect the quality of the code of an application, and as such an application with a bigger attack surface does not necessarily mean it has many vulnerabilities and vice versa, but instead, a larger attack surface measurement reflects that an application is likely to be exploited with little effort and cause more damage to it.[11] So, does the quantity of attack surface parameters increase the attack surface of an application, or only a few have a significant influence towards the size of the attack surface of an application? Which attack surface parameters have a greater contribution to the size of the attack surface of an application? Does incorporating all attack surface parameters at once in an application make it have a bigger attack surface than having a selected few?

To answer all the questions above, we compare the attack surface of Glastopf by using four different web templates. Our main focus is to find out how much metrics does the contents of a web template brings to the overall attack surface of the application. First, we created a simple html page, supplied it to Glastopf and then measure its attack surface. By definition, this setup should return the smallest attack surface mainly because the templates contain almost nothing that contributes to the size of the attack surface. A simple and pure html page with text and images only as its content will produce a small attack surface than a page with more parameters such as forms, dynamic content, cookies, etc. as its contents. The original Glastopf templates are purely dynamic. A script fetches data from a text file and populates the template with that text. However, there is presence of forms in the original templates which according to Goswami et al.[6] and Manadhata and Wing [12] contributes to the size of the attack surface. At this moment, we already know that two attack surface parameters will have an effect on the size of the attack surface of Glastopf, but by how much is not really known. Thus, the difference in the values we get from the plain template and the original Glastopf templates should reflect how much attack surface metrics the forms and dynamism brought to the application.

The third template contains the attack surface parameters as defined by Goswami et al.[6] and Manadhata and Wing.[12] All the attack surface parameters are included in the template and then its attack surface measured. The idea is to find out which parameters contribute most to the size of the attack surface of the application. Since the objective is to increase the attack surface of the application, once the parameters that contribute most to the size of the attack surface are established, then the best possible ways of deploying those parameters in a template would be determined.

Thus, do we have to duplicate the attack surface parameters in our application, or does duplicating parameters lower the attack surface measurement, or it has no effect at all?

The last template contains all the parameters selected in the previous approach (template 3), but now they are implemented in a content management system (CMS) template. This approach tries to increase the attack surface of an application by bringing content-based package dependency into play. Vulnerabilities on highly dependent packages usually bring larger attack surfaces compared to those detected on a client application, even when they have the same CVSS scores,[13] and CVSSv2 does not reveal that fact. Thus, by bringing CMS content-based package dependency vulnerabilities to Glastopf, the attack surface of this application would increase even though that may not reflect in CVSSv2 score. However, for the use of CMS to be successful in this application, there is a need for interfacing Glastopf with wordpress, which will be discussed in our next paper.

Observations

In this research, we use attack surface parameters as suggested by Heumann, Türpe, and Keller[5] and Goswami et al.[6] to find out by what capacity they contribute to the attack surface. Our goal is to maximize the size of our attack surface on a dynamic web application honeypot such as Glastopf, thereby increasing its chances of gaining more attention from the attackers. However, the degree of distribution attack surface parameter was not tested in our application. This is neither because it is of little significance to the size of the attack surface, but rather, because our experimental setup was not in a public domain where such distribution setup can be tested. We were also unable to test access control parameter because our application can only be accessed on privileged rights, thus sudo rights. It is worth noting that the measurement shown by the scanner does not reflect the number of vulnerabilities in an application but rather the ease or difficulty at which the application can be exploited. The bigger the number reflected by the scanner, the easier the application is to exploit.

Using Nessus vulnerability scanner to measure our application's attack surface, our plain html template produced the smallest attack surface, and by definition, it was the expected results. A total of 71 were shown by the scanner. However, Glastopf has several vulnerabilities which it emulates, so from this reading and looking at the vulnerabilities shown by the scanner, we can safely say the majority of this attack surface measurement was coming from Glastpof's emulated vulnerabilities rather than the template itself.

On the plain html template above, we then introduced one attack surface parameter, the form, and measure the attack surface again. The presence of security mechanisms in an application reduces the attack surface of the application. Since our goal is to in-

crease the attack surface of Glastopf as much as possible, we then created our form without any input validation or security mechanism. The attack surface of the application now increased to 74, which means the form, increased the attack surface by a value of 4. We then created another form in the application, without any input validation or security mechanism just like the first form. This was to test if duplication of the same attack surface parameter affects the size of the attack surface. The reading in the scanner didn't change, thus, the attack surface measurement remained at 74. From this test, we can safely say duplication of the same attack surface parameters in a single web template does not affect the attack surface of an application.

In the next test, we created our template using the Jinja2 templating language. By doing this, we have now transformed our plain html template into a fully dynamic web template. The reading from the scanner was now at 102. Thus, the dynamism of the template has now increased the attack surface of the application by a value of 28. This reading was also achieved when scanning Glastopf with its original templates. So, from this test, we can safely say that the original Glastopf templates has only 2 attack surface parameters, which is presence of forms and dynamic page creation. Secondly, we can conclude that dynamic page creation increases the attack surface by a bigger magnitude than just html forms.

In the next test, we then introduced cookies in our dynamic web template. First, we created static cookies in the template. A static cookie is the one whose cookie data is not changeable, and is mostly created from html only. The purpose of a static cookie test was to find out if dynamism in the cookie will affect the results as it was the case with the webpage template and by how much. In this test, our scan results value increased by 2. We then created a more complex cookie by making it dynamic using jScript. The scan results showed that the value increased by 4 from the previous test. From this test, we can safely say the more complex the cookie is; the bigger it contributes to the size of the attack surface on an application.

Another parameter that is known to increase the size of the attack surface of an application on the client side is active content. Active content basically refers to interactive or dynamic content in a webpage, which include programs such as javaScript applications, activeX aplications, animated images, stock tickers, internet polls, action items, streaming video and audio, weather maps, embedded objects, and many more. In this test, we first created an embedded object; a flash object in our template, and then scan the surface. The scanner recorded an increase in the size of the attack surface by a value of 2. Then we embedded a streaming YouTube video into our web template and measure the surface. The objective was to find out if having different active content in the page would increase its attack surface as opposed to having just one. The results did not change from the previous test. We then included a poll, first

without submitting any information to the database, and then changed our poll to start submit the poll results to the database. However, the database has been pre-existing to the application, but we just modified it to store the poll results. The size of the attack surface did not change from the previous test when it is not submitting the poll results to the database, and increased by 3 when it is submitting to the database. On the last test in active content, we created a news feeds in the template and then scan the surface, and the size of the attack surface increased by a value of 5.

The diagram below shows a summary of how different attack surface parameters affect the size of Glastopf relative to the original or un-edited Glastopf web application honeypot.

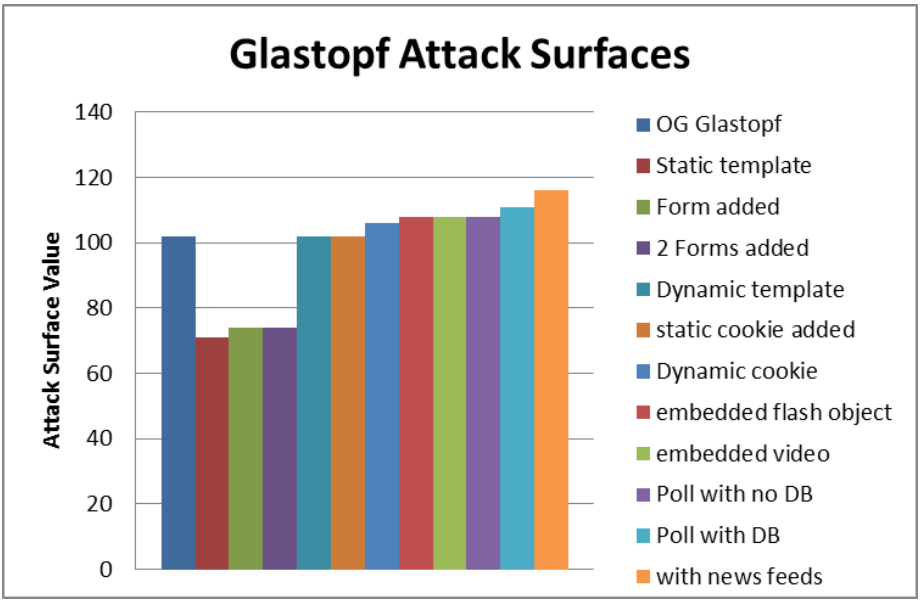

Figure 2: Effects of Attack Surface Parameters on Honeypot.

Discussion

This research sets out to explore how one can maximize the size of an attack surface of a dynamic web application honeypot so that it becomes more interesting to the attackers out there looking for what to attack. In particular, the research uses the attack surface parameters that have been identified to have an impact in the size of the attack surface of an application.[5, 6] Thus, how can one use attack surface parameters to get the biggest attack surface in a web template? The attack surface of a web applica-

tion is not only contributed by the web template but the entire application itself. In this research, we were concerned with how the web template impact on the size of a web application attack surface if these parameters are deployed in a particular way.

So, the question may be: what is a large or a small attack surface of an application? In our case, we measure our attack surface relative to the original Glastopf's attack surface. In formal descriptions,[14] says an attack surface of system A is larger than that of system B given that

An environment $E_{AB} = \langle U, D, T \rangle$, *the attack surface* $\langle M_A, C_A, I_A \rangle$, *of a system A, is larger than the attack surface* $\langle M_B, C_B, I_B \rangle$, *of a system B, iff:*

i. $M_A \supset M_B \wedge C_A \supseteq C_B \wedge I_A \supseteq I_B$

ii. $M_A \supseteq M_B \wedge C_A \supset C_B \wedge I_A \supseteq I_B$

iii. $M_A \supseteq M_B \wedge C_A \supseteq C_B \wedge I_A \supset I_B$

Where U,D,T are elements of a system's environment to mean the user of the system, the data stores of the system, and the communications channels of the system, and M,C,I are the elements of an attack surface of a system to mean the entry and exits points of a system, set of channels used by the system, and untrusted data items in the system. In our case, the attack surface elements are the actual attack surface parameters we used to model our system's attack surface.

The above proposition says the attack surface of system A is bigger than the attack surface of system B if and only if; i) the entry and exit points of system B are a subset of system A where else the communication channels and untrusted data items of both systems are the same, or, ii) the set of channels use d by system B is a subset of channels used by system A where everything else is the same, or iii) the untrusted data items for system B is a subset of untrusted data items for system A where everything else is the same.

Krautsevich et al.[11] give another formal description on the size of attack surface:

Let X_A *be a set of attackers relevant for system A, and* X_B *be a set of attackers relevant for system B. We say that system A is more secure than or equal to B* $(A \geq_s B)$ *if* $\Gamma_A \subset \Gamma_B$, *where:*

$$\Gamma_A = \left\{ \gamma' x \middle| \gamma' x \in X_A \in X_A \; \gamma = \gamma' \bullet \gamma' x \; \Lambda \; A \| X_A \xrightarrow{\gamma} A' \| X'_A \Rightarrow P_{sec}(A' \| X'_A) = T \right\}$$

$$\Gamma_{B=}\left\{\gamma'' x \middle| \gamma'' x \in X_B \in X_B\ \gamma = \dot{\gamma} \bullet \gamma'' x \wedge\ B \| X_B \xrightarrow{\dot{\gamma}} B' \| X'_B \Rightarrow P_{sec}\left(B' \| X'_B\right) = T\right\}$$

The principle above says that if a set of possible attacks on one system is larger than a set of attacks on another system, then the later system is more or at least equally secure than the former one. In our tests, we deployed our attack surface parameters in different arrangements, and also introduced attack surface parameters that were initially not present in Glastopf. From these tests, we found that deploying two identical attack surface parameters in a web application does not affect the size of its attack surface. If for example, you have a form in your web template, having another form of the same type would not increase nor decrease the attack surface of your application. In the table below, we outline an algorithm that one can use to decide whether to include a certain parameter in the application web template.

Table 1: Adding Parameters to the Template to Increase the Attack Surface

Algorithm 1: Add parameter to the template	
1: Function Add_Parameter (bolean exist)	
2: While exist == true {	
3: If ($pA \supseteq pB$)	*: A is a superset/ includes B*
4: Then ($pB \geq_s pA$)	*: B is more secure than A, therefore won't increase AS*
5: Add_Parameter() = false	
6: Else Add_Parameter()	
7: If exist == false	
8: Add_Parameter()	

The algorithm says, when you decide to include a parameter in the web template of an application so as to increase its attack surface, first check if that parameter does exist. In the table above, we assume that pA is the existing parameter and pB is the parameter to be added to application template. If the parameter does exist, check if the functionality and the elements of that functionality are the same as the existing parameter. Line 3 in the algorithm says if parameter A is a superset of parameter B, then parameter B is more secure than parameter A (line 4), therefore parameter B will not increase the attack surface of that application hence no need to include in the application. If the parameter doesn't exist in the application (line 7), then adding it would automatically increase the size of the attack surface.

Our research also found that different attack surface parameters increase the size of the attack surface with varying degrees. For example, in our research we found that the dynamic web template design has a bigger effect in the size of the attack surface compared to all other parameters tested. These suggest that when the aim is to in-

crease size of the attack surface of a web application, having a dynamic web template would be an obvious choice over a static template. The more complex the application becomes the bigger the attack surface it would have. By adding an attack surface parameter that is either not existing to the application template or it is existing but has either different functionality or different elements would always increase the size of the attack surface of an application. But increasing the complexity of that functionality by either adding elements such dynamism and third party interactions will increase the attack surface further.

References

1 Iyatiti Mokube and Michele Adams, "Honeypots: Concepts, Approaches and Challenges," Conference Proceedings of the 45th Annual Southeast Regional Conference (Winston-Salem, North Carolina, USA, March 23-24, 2007): 321–326, https://doi.org/10.1145/1233341.1233399.

2 Lance Spitzner, *Honeypots: Tracking Hackers* (Boston, MA: Addison Wesley, 2002).

3 Mathias Gibbens and Harsha vardhan Rajendran, "Honeypots," (April 2012), 1–12., available at https://www2.cs.arizona.edu/~collberg/Teaching/466-566/2012/Resources/presentations/2012/topic12-final/report.pdf.

4 ENISA, "Proactive Detection of Security Incidents – Honeypots," (2012): 181.

5 Thomas Heumann, Sven Türpe, and Jörg Keller, "Quantifying the Attack Surface of a Web Application," in *ISSE/Sicherheit 2010: Information Security Solutions Europe*, Sicherheit, volume P-170 of Lecture Notes in Informatics (LNI) (Bonner Köllen: Verlag, 2010): 305-316.

6 Sumit Goswami, Nabanita R. Krishnan, Mukesh Verma, Saurabh Swarnkar, and Pallavi Mahajan, "Reducing Attack Surface of a Web Application by Open Web Application Security Project Compliance," *Defence science journal* 62, no. 5 (2012): 324–330, https://doi.org/10.14429/dsj.62.1291.

7 Klaus Berberich, Michalis Vazirgiannis, and Gerhard Weikum, "Time-Aware Authority Ranking," *Internet Mathematics* 2, no. 3 (2004): 301-332.

8 Peter A. Hamilton, "Google-bombing – Manipulating the PageRank Algorithm," *CMSC 676 - Information Retrieval* (2013): 1–5, available at https://pdfs.semanticscholar.org/2bce/4885f4d27923acc283af760027cec94ccbdf.pdf.

9 Ricardo Baeza-Yates, Icrea-univ Pompeu Fabra, and Carlos Castillo, "Pagerank Increase under Different Collusion Topologies," in Proceedings of the Workshop on Adversarial IR on the Web (2005), 25-32, available at http://airweb.cse.lehigh.edu/2005/baeza-yates.pdf.

10 George Markowsky, "The Metric at the End of the Rainbow," in The TIEMS USA 2017 Annual Conference Emergency Management, Homeland Security, and Computing, 2017.

11 Leanid Krautsevich, Fabio Martinelli, and Artsiom Yautsiukhin, "Formal Approach to Security Metrics. What Does 'More Secure' Mean for You ?" Proceedings of the Fourth European Conference on Software Architecture, Copenhagen, August 23-26, 162–169, available at https://doi.org/10.1145/1842752.1842787.

[12] Pratyusa K. Manadhata and Jeannette M. Wing, "An Attack Surface Metric," *IEEE Transactions on Software Engineering* 37 no. 3 (May-June 2011): 371–386.

[13] Su Zhang, Xinwen Zhang, Xinming Ou, Liqun Chen, Nigel Edwards, and Jing Jin, "Assessing Attack Surface with Component- Based Package Dependency," in *Network and System Security*, Lecture Notes in Computer Science, vol. 9408 (Springer, Cham: NSS, 2015), https://doi.org/10.1007/978-3-319-25645-0_29.

[14] Pratyusa K. Manadhata and Jeannette M. Wing, "A Formal Model for A System's Attack Surface," in *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats* (2011): 1-29, https://doi.org/10.1007/978-1-4614-0977-9_1.

About the Authors

The author of this article are with the Botswana International University of Science and Technology and the Botswana Institute of Technology, Research and Innovation.