Study on Web Security Issues

G. Baby
Department of MCA
KMM Institute of PG Studies
Tirupathi, India.

M. Sethu Ram
Asst.Professor
School of Eng&Tech,
SPMVV,Tirupathi, India

Prof. M. Padamavathamma
Department of Computer Science
S.V.University
Tirupathi, India

Abstract: ***Managing*** **Web Application Security for an organization that engages in electronic business is a huge and complex task. Every entry point in the e-Business system must be secured, at both the network and application levels. Most of the network security issues, including access control, data transmission security, and authentication can be addressed using commercially available products, but application security has received less attention. Consequently, the application has remained the most vulnerable component in the security chain. Today, almost every e-Business web site can be broken into the application level in a matter of hours. Hacking techniques such as hidden field manipulation, parameter tampering, and cookie poisoning can be easily deployed, resulting in stolen customer data, denial of services, and the complete shut-down of the site. The vulnerabilities that permit these exploitations exist as a result of flaws in the design, implementation, and testing of internally developed code, as well as bugs and the misconfigurations of third-party products. It describe a new security technology, Web Application Shielding, a runtime application-level security proxy that automatically recognizes the application security policy for each page by constantly analysing the outgoing traffic from the web application to its clients. The proxy then automatically enforces this policy on returning requests, preventing hackers from exploiting application vulnerabilities and removing the need to track and patch every hole in the application. This not only provides a higher level of security, but also reduces security resource requirements within the organization.**

Keywords: Web application, ROI, Web security, Web security approaches.

1. INTRODUCTION

Most of the business organizations in this current technology era depends on online software applications to conduct their operations. For this reason, web applications are subject to various types of attacks as hackers exploit vulnerabilities within the software like SQL Injection and XSS [1]. Malicious or silly users are also responsible for web security problems like users misuse the application to cause system failures or denial of service attacks, or attempt to discover web application vulnerabilities. To detect web-based attacks, intrusion detection systems (IDSs) are the most popular solutions to detect the web attacks. Unfortunately, it is hard to detect web application attacks due to the following two reasons. First, web applications are so specific and even the same applications might be deployed in different platforms, coded by different programmers or implemented by different programming languages. Second, IDSs must keep the signatures updated with respect to the large numbers of vulnerabilities discovered daily.

It is going to raise a problem relating to system performance. Thus we cannot use the same signatures to defend the same attacks to different websites because the website vulnerabilities result from different software errors and programming skills. This paper presents an on-line user-behaviour surveillance system that detects the malevolent user behaviour and web application attacks using Embedded Markov model (EMM). The EMM proposed in this paper is a two-phase Markov model [3]. The first phase Markov model is used to establish the normal model for every attribute. The second phase is used to build the user–behaviour surveillance model for detecting the abnormal visiting behaviour. Our experiments show that our on-line user behaviour surveillance system is immediately useful to strengthen the on-line operating web security.

A. WEB SECURITY APPROACHES

The following approaches can be followed to add security to online web applications.

- ✓ **Bolt-on approach**
- ✓ **Do-it-all-up-front approach**
- ✓ **Big-bang approach**
- ✓ **Buckshot approach**
- ✓ **All-or-nothing approach**

Now we are mainly concentrated on working approaches, this approaches is used protect the security for the web applications.
So if bolting security on to your application or taking a buckshot approach isn't producing effective results, what works? The answer is a life-cycle approach, or "baking" security into your application's life cycle. By leveraging it in the development life cycle, you address security throughout development instead of up front, after the fact, or, worse, randomly. Infusing security into the life cycle is also a practical way to balance security with other quality attributes, such as performance, flexibility, and usability.

B.ROI Activities

Clearly Return-on-investment (ROI) activities should be formed and applied entire development life cycle as a baseline set. For each of the scenario ROI activities be supplemented or modified accordingly. Most of the companies have some variations of the following activities, independent of any security focus:

• Design guidelines,
• Architecture and design review,
• Code review,
• Testing, and
• Deployment review.

Identifying design guidelines involves putting together a set of recommendations for the development team. These recommendations address key engineering decisions (such as exception management) and include input from software vendor recommendations, corporate policies, industry practices, and patterns. It's a high ROI activity because it helps create the project's scaffolding.

Architecture and design review is an exercise in evaluating the design against functional requirements, non-functional requirements, technological requirements, and constraints. It can be as simple as a white- board sketch, or it can involve multiple documents, diagrams, and presentations. An architecture and de- sign review is most valuable when it's performed early enough to help shape your design. It's less valuable when it's performed so late that it's only function is to find "do overs."

Code review is a practical and effective way to find quality issues. Although you can find some issues through static analysis tools, the advantage of a manual code re- view is contextual analysis—for example, you know the usage scenario and likely conditions.

Testing is executable feedback for your software: it either works or it doesn't. Whether it works "right" is a bit trickier to establish. The ideal case is the one in which you establish an executable contract between your code and requirements, including functional, non-functional, and constraints. It also helps to know what you're looking for so you can test against it. Although testing is usually optimized around functional behaviour, you can tune it for quality attributes depending on your ability to set measurable objectives and de- fine what "good" looks like.

Deployment review is an evaluation of your application as it's deployed to your infrastructure—where the rubber meets the road. This is where you evaluate the impact of configuration settings against runtime behaviour. A deployment review can be your last actionable check- point before going into production. Preferably, you have a working build of the software early in its development cycle so that early deployment reviews can help reduce surprises going into production.

These activities offer a high ROI because, if applied properly, they directly affect the software's shape throughout the process. I've seen these activities transform my customers' results for the better, and they pro- vide key feedback and action at high-leverage points in the development process.

III SECURITY ENGINEERING BASELINE ACTIVITIES

Effective security activities based on the high ROI activities must be listed. However, rather than interspersing security in your existing activities, factor it into its own set of activities. This will help optimize your security efforts and create a lean framework for improving your engineering as you learn and respond. To influence software security throughout the development life cycle, you can overlay a set of security-specific activities. Table 1 shows the key security activities in relation to standard development activities. The following is a set of baseline activities for security engineering:

• Objectives,
• Threat modelling,
• Design guidelines,
• Architecture and design review,
• Code review,
• Testing, and
• Deployment review.

These activities complement one another and have a synergistic effect on your software security.

A. *OBJECTIVES*

To perform any job, you need to know where to start and when you're done. The security objectives activity is an explicit activity that helps you set boundaries and constraints for your software's security so that you can prioritize security efforts and make any necessary trade-offs. Security is a quality attribute that you must balance with other quality attributes and competing concerns, as well as project constraints. You should identify up front how important security is for your particular project and context. Identifying security objectives helps you identify when you're done from a security perspective. A starting point might be to state what you don't want to happen with respect to security—for example, you don't want to leak customer data or for one user to see an- other user's data. For this activity, it's important to first agree what is inside the scope of your information security. It helps to think in terms of the information's confidentially, integrity, and availability (CIA). You next decide which categories of objectives to work with. Using categories such as client data, compliance, quality of service, and intangible assets helps here.

The following guidelines can help you identify your own security objectives:

- Use scenarios to set bounds. Are operational requirements out of scope?
- Ask yourself, "What's the worst thing that could happen from a security perspective?" This helps you identify how important security is and where to spend more energy.
- Turn objectives into requirements and constraints.

Conversely, you can avoid common mistakes when identifying security objectives:

- Don't try to figure out all your security objectives up front. What matters is that you identify the important ones that could have a cascading impact on your design.

B. *THREAT MODELLING*

We are familiar with your system's threats so that you know whether your security mechanism is effective or even warranted. Threat modelling, an engineering technique you can use to shape your software design, will help inform and rationalize your key security engineering decisions. In its simplest form, a threat model is an organized representation of relevant threats, attacks, and vulnerabilities to your system. During design, the most effective threat-modelling technique lets you quickly play out various "what if" scenarios [2]. Long before you implement an expensive countermeasure, you can quickly evaluate whether you're even going down the right path. You can certainly use threat modelling on a deployed system, but you'll likely find overlap with existing risk assessment activities. Risk assessments typically focus on the business risk, and often have an operational or infra- structure focus, so they're not usually optimized for direct usage by the software engineer. In practice, I've found the most successful threat modelling activities partition the infrastructure threat models from the applications.

The following guidelines can help you build a threat model:

• Incrementally render your threat model. Model it to identify what you know, what you don't know, and what you need to know next.

• Use a question-driven approach. Whether you're figuring out threats, attacks, or vulnerabilities, asking the right questions can lead you to better answers. You can even seed your questions with lists of questions that experts ask.

• Use a threat model to identify what code to prototype and test. Your threat model will likely reveal high-risk engineering decisions, which are great candidates to prioritize what to prototype or spike.

• Use complimentary techniques. You can use various techniques for finding threats, attacks, and vulnerabilities, such as use-case analysis, dataflow analysis, question-driven approaches, roles matrices, and brainstorming.

• Use pattern-based frames. They're a conduit of insight from expert to practitioner and can significantly help find the low-hanging fruit as well as organize your thinking.

• Partition your threat-modelling activity within a predetermined time period. threat models. For some companies, partitioning by network, servers, desktops, and applications helps optimize relevancy and ownership. The key is to identify the important intersections between the application and infrastructure.

C. DESIGN GUIDELINES

Is your development team using proven software security design principles, practices, and patterns? If you haven't pulled some together, you have less of a chance of success. The security guidelines activity is where you hunt and gather relevant security design advice and organize it in a way that's useful for your development team. This can include vendor recommendations and home-grown experience. A good guideline, however, will be more than just a description; it will be prescriptive. "Use input validation" is clearly a good idea, for instance, but it isn't a good guide- line. Let's turn that nice idea into something we can act on:

• What to do: Don't rely on client-side validation. Use it to help reduce the number of round trips to the server, but don't rely on it for security.

• Why: Serve side code should perform its own validation because an attacker might bypass your client or shut off your client-side script routines.

• How: Use an "allow" approach over "deny." It's un-likely that you can define all the possible variations of what bad input would look like. Instead, define what good input looks like and filter for that.

D. SYSTEM ARCHITECTURE

Generally user behaviour surveillance system starts by extracting the different user visiting sequences from web traffic. Then, the analysis focuses on GET and POST requests. Our system architecture can be divided into two parts as shown in below Figure. One is user behaviour surveillance modelling system and the other is online user-behaviour surveillance defending system. We collect the specific website traffic and then process the user visiting data on the user behaviour surveillance modelling system. User visiting page transition, user query attribute values and all accessed web pages will be extracted from the web data and fed into EMM training model.

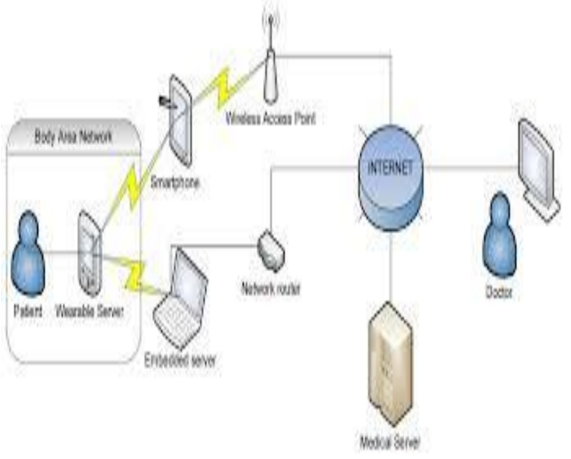

Fig: 1 EMM Training Model

The accuracy of all attributes for each page and the user visiting page transition sequences will link up to complete the user behaviour surveillance model. In the on-line defending system, we implemented the software which was called the behaviour supervisor. The behaviour supervisor will monitor the user visiting behaviour and coordinate with user behaviour surveillance model to defend the user abnormal behaviour. When the behaviour supervisor finds the user abnormal behaviour, it will block the user's connection and record the user's information. Moreover, the normal traffic will be stored into real traffic collection database that will support our user behaviour surveillance model to be retrained. The retraining will gradually increase our system's accuracy and intelligence.

E. CODE REVIEW

It's important to keep in mind that some security choices are about following building codes or proven practices, while other choices are more about business risk. For example, following a vendor recommendation for a coding practice is closer to a building code, which might take the form of compliance rules, vendor recommendations, or your own policies. Other choices might be primarily driven by business risk. When you're faced with a few ways to solve a problem, in the absence of a clear coding practice, business risk is usually a driving factor. Many security practices become a combination of business risk and proven practice. At the end of the day, it's your security objectives that help you know where to set the bar and how far to go Security experts use four common steps to per-form security code reviews:

1. Identify security code review objectives. This is where you establish the review's goals and constraints.
2. Perform a preliminary scan. This involves using static analysis to find an initial set of issues and improve your understanding of where you will most likely find is- sues when you review the code more fully. The key here is to not get lost in the potential noise and false positives, but to see patterns of where to look further.
3. Review code for security issues. This is where you review the code thoroughly to find security vulnerabilities. You can use the results of Step 2 to focus your analysis. Here, techniques such as control- and dataflow analysis come into play.
4. Review for security issues unique to your architecture. This step is most important if you've implemented a custom security mechanism or any feature designed specifically to handle known security issues.

F. TESTING

A lot of testing is requirements-based because requirements help define an application's scope. The problem is that requirements are more about functionality than qualifiers (such as what "good enough" security looks like). With performance, you can measure response time, throughput, and resource utilization. With security, it's difficult to measure confidentiality, integrity, and availability. The bar for knowing where to start, how to proceed, and when you're done goes back to your security objectives.

The most effective, efficient security testing revolves around your security objectives and threat model. This doesn't dismiss possible benefits from exploratory testing or discount penetration testing. In fact, it's an industry practice to get a third-party to assess your software security when the stakes are high, which often includes penetration testing, both white and black box.

G. DEPLOYMENT REVIEW

Deployment is when your application meets the infra-structure, so deployment review is a great opportunity to assess the final end game. Hopefully, production isn't the first time you introduce your application to your deployment scenario. In practice, this often seems to be the case because I continue to hear horror stories in which an application roll out went bad or got blocked for reasons that an earlier deployment review would have caught.

If you have a lot of knobs, switches, and desired set- tings or states, categories can help. A flat list of settings can be overwhelming, and it's tough to rationalize conflicting lists, particularly from different sources with completely different lists. I like to organize the options by actionable categories. You might name a category "Ac- counts," for example, and then identify the end state, based on principles:

• Accounts are least privilege,
• Strong password policies are enforced, and
• Separation of privileges is enforced.

You can then quickly normalize compliance rules, infra-structure constraints, vendor recommendations, and so on. This also helps make checking more repeatable during deployment reviews.

Conversely, you can avoid a common mistake: don't depend solely on your checklists. They're a great way to catch low-hanging fruit, but they don't replace thoughtful, contextual analysis.

IV CATEGORIES IN THE WEB APPLICATION SECURITY FRAME:

The following should be considered scrupulously for a robust application.

• Input and data validation,
• Authentication,
• Authorization,
• Configuration management
• Cryptography,
• Sensitive data,
• Exception management,
• Session management, and
• Auditing and logging.

Web application security is same as software security model with properties and attributes is a vital component in successful project development, often does not include sufficient attention to security concerns. The security considerations and issues must be addressed with application design, development, deployment, and maintenance in view, not during any one of these phases in isolation [4].

V CONCLUSION

The web application security advancements that we have discussed in this article without a doubt raise the bar in securing web applications by allowing web developers to establish an additional layer of defence as well as remediate the common web application vulnerabilities efficiently. While web developers are strongly recommended to investigate and leverage these techniques within their applications it should also be noted that the modern browser is a key component in the enforcement of these techniques. Hence these new techniques don't serve as a complete replacement to the existing remediation techniques since users not using the modern browsers would have no protection against the vulnerabilities that they remediate. The expert services and education to help organizations continuously and measurably protect their most important assets from the most critical threats. Through a strategic approach to security, Founds stone identifies and implements the right balance of technology, people, and process to manage digital risk and leverage security investments more effectively.

VI REFERENCES

[1] C. Kruegel, G. Vegan, “Anomaly detection of web- based attacks”, Conference on Computer and Communications Security, Proceedings of the 10th ACM conference on Computer and communications security, pp.251-261, 2003**.**

[2] “Threat Modelling Web Applications” at http://msdn. Com/ThreatModeling.

[3] P. Zhong, J. Chen, *A Generalized Hidden Markov Model Approach for Web Information Extraction*. IEEE/WIC/ACM International Conference on Web Intelligence, pp.709-718, 2006

[4] D. Brumley, J. Newsome, D. Song, H. Wang; S. Jha. *Towards automatic generation of vulnerability-based signatures*. In Proceedings of the 2006 IEEE Symposium on Security and Privacy, pp.15-, 2006.