Evolution of Web Security - A Discussion mainly based on the Updates of OWASP TOP 10

Siwei Xie

Univeristy of Stuttgart
st165900@stud.uni-stuttgart.de

Abstract. Web service is widely used as a solution that realizes fast and flexible information sharing between customers and companies. As a modern human being, it is inevitable to use web-based applications and allow them to obtain some important sensitive information within the scope of access control. As a result, these web applications also inevitably become the preferred targets for attackers on the internet and the emerging threats have brought huge challenges to web security. This seminar work mainly discusses the evolution of common web attacks from Web 1.0 to Web 2.0, after that, the potential risks in Web 3.0 and future web will be discussed, which is mainly based on OWASP top 10 lists (in total 7 updates so far, in 2003, 2004, 2007, 2010, 2013, 2017, and 2021 respectively). Meanwhile, from the perspective of a developer, it is also meaningful for us to cultivate the awareness of "secure programming" from the basics.

Keywords: Web security · Web attacks · OWASP TOP 10 · Broken Access Control · Injection · Insecure Design

1 Introduction&Background

1.1 Insecurity by Default

Why are websites or web applications often easily under attack? The answer is that web is insecure by default. By definition, web-based applications are client-server programs that can be accessed over the web and the HTTP protocol[1]. More technically oriented, users trigger a request to the web application server through the browser, then web application server manages the database server in order to complete the requested tasks, eventually the browser runs all the responsible scripts and displays the results to the users. Because of its platform-independence, less needed processing power, higher compatibility and easier update, web applications have become a popular replacement of desktop applications in the past decade. Compared with desktop applications, they don't take up space on the hard drive, but store the files on a remote server, so that more points of entry are available as long as the internet connection is reliable. Then it is unfortunate but inevitable that the security of web applications are sacrificed for higher accessibility. When any sensitive information is

located remotely and can be exchanged anytime, which means the end-users cannot directly manage their valuable data storage but rely on cloud or on-premise services for authentication and authorization, the privacy and security problems arise.

Although countermeasures are taken to increase the security level, but the fact is that it's impossible to eliminate vulnerabilities completely and risk is still everywhere. The use of firewalls for preventing unwanted access and the use of SSL in transport layer for protecting messages are helpful. Those unnecessarily exposed ports and unauthorized access should be stopped by taking these protective measures, but meanwhile the specific ports for accessing database must be made explicit in order to provide corresponding services, and this basically gives intruders the chance to communicate with them successfully if they can elude the security mechanisms provided by the operating system. Except for the potential web attacks from outside, the threats from inside should not be ignored as well. If there is no access control between the server and intranet, hackers will have interest in intrusion into those terminals with much less protection, although those valuable data are not located there, through them web service can be damaged from inside and finally hackers can still achieving their goals. It is also noticeable that web applications are made up of programs, so they definitely contain bugs and flaws which can be maliciously used for stealing important information. Insufficient programming practice and coding errors always lead to less rigorous input and result validation, which is a glaring weakness that hackers try to find in reality. For example, avoiding direct database access but using web service with restricted access to the database tends to be safer, but security leaks related to authentication, availability and integrity can still be captured by attackers if the enforcement of responsible web service operations is not perfectly correct set up by developers.

Moreover, a more practical reason is that web technologies and accompanying attacks are evolving quickly, what is secure now may be insecure soon. Because of higher complexity and higher investment of implementation, it's hard to think about security in the whole software development life cycle(SDLC). "More secure" can be products' competitive advantage, but due to the disproportionate correlation between input and output, it is always seen as the lowest priority.

1.2 Sketch the Evolution of Web Attacks

The evolution of web attacks can be divided into several stages depending on the development of the web itself.

The first generation of Web, namely Web 1.0, was quite limited in use with only simple information retrieval functionality. HTML, URL and HTTP specifications construct the core architecture of Web 1.0[1]. In that period, websites didn't have highly complicated running logic. The main content of websites was static and table-based, and the user behavior was also limited, that is, simply browsing the

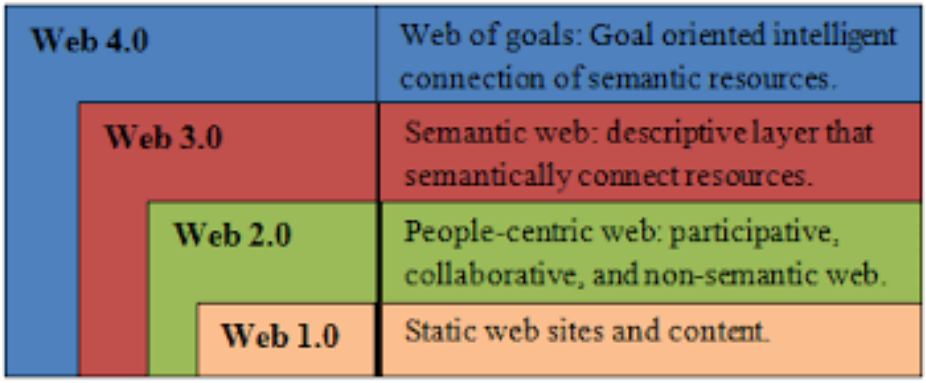

Fig. 1. The Evolution of Web[7]

web. Back then, there was more security concern about the dynamic scripts on the server side, such as uploading a web shell to the remote server in order to gain access. Later in this period, SQL Injection first appeared in 1999, which can be reckoned as a milestone in the history of web security. Hackers found that they could get a lot of important data through SQL Injection, and some even gained access to the server system. Gradually it becomes the most popular means of web attacks. Then came cross-site scripting(XSS) attacks. Actually it appeared almost at the same time as SQL Injection, but then security focused more on SQL Injection, until 2003 people realized its huge attack power due to the well-known Myspace XSS worm.

After 2005, people entered the Web 2.0 era, and a large number of web applications were created. The front-end has undergone a radical change. Web pages no longer simply consist of pictures and texts because they don't meet users' requirements anymore, and the diversified interaction and intimate involvement also brings a fresh experience to users. The most typical businesses in Web 2.0 are internet banking and e-commerce platforms. Because the interaction of webpages was greatly enhanced and the back-end used a large number of databases for storing sensitive data, hackers aroused interests in making profit from them. At that stage, the outbreaks of web security problems attracted extensive attention, a variety of SQL Injection, XSS, unauthorized access and other kinds of web attacks included.

In Web 2.0, enterprises mainly provide as much content and information as possible for users to use, while in the privacy-driven present and future based on Web 3.0, namely the semantic web, with the support of big data, artificial intelligence and IoT, the contents that enterprises actively push to users for marketing become more accurate. The contents, scenarios and ways of interaction between users and back-end servers will be greatly expanded, and since the user needs change frequently, the user experience and various personalized needs of

users in that stage are widely satisfied by a large number of various applications. At the same time, user information becomes the most valuable data. Without them, plenty of functions mentioned above cannot be implemented. The rise of script languages, automated operation and maintenance methods, and AI technologies, apparently lower the threshold of becoming a hacker and expand the scope and efficiency of stealing information. As for now, it seems that those new technologies are more helpful for hackers. In particular, currently there is a huge amount of leaked database information on the internet, and hackers can use various automated hacking tools to obtain user information in a more relaxed and relatively safe way.

1.3 OWASP & OWASP Top 10

The OWASP Top 10 is an open-source research project which is aimed at providing rankings and prevention recommendations for the top 10 most serious web application security risks[8]. According to the type and severity of vulnerability, the total numbers of individual security breaches and their degree of danger, risks are categorized and placed. The purpose of this project is to warn web application security professionals and developers about the most common security risks, so they can make use of the report's findings to defense their software. OWASP manages the Top 10 list and began since 2003. They update the list every 2-4 years to keep up with updates in the AppSec market.

The evaluation about the evolution of web attacks are mainly based on the updates of OWASP top 10 lists. Depending on when they were released and the equivalent evolution of web, they were roughly divided into three stages for interpretation. There is no clear and strict boundary between each stage, as the term "Web X.0" was come up for describing characteristics that meet user requirements at different periods and guided the development direction of web technologies. Though evolving technology stacks bring more up-to-date user experience and gradually take a larger share of the market, it doesn't mean that they can totally replace the preceding ones, just keep abreast of the growing trend and let them coexist. A persuasive example are web applications that play a role of encyclopedia nowadays, such as Wikipedia, static pages and multiple hyperlink jumps are very typical Web 1.0 approaches, but participation and contribution from user side are more like Web 2.0 features. In fact, it is impossible to separate Web 1.0, Web 2.0 and Web 3.0 very precisely in the timeline and each innovative concept is a gradual expansion of the previous framework.

The grouping for discussion is as follows: the first two published findings in year 2003 and 2004 corresponds to the late stage of Web 1.0, and the next four published lists in year 2007, 2010, 2013 and 2017 respectively will demonstrate the common security risks in the on-going Web 2.0, finally the recent update in year 2021 compared with the previous update is for illustrating and predicting security concerns in the approaching Web 3.0 and future Web.

2 WEB 1.0 (OWASP TOP 10 in year 2003 and 2004)

2.1 Web 1.0: read-only static web

Web 1.0 is the first generation of the World Wide Web evolution, invented by Tim Berners-Lee. Websites were "informational" and only allowed one-way communication. There were a small number of producers and a huge majority of consumers who can only read information but can not do any own contribution to the content of the pages. Web 1.0 application is a interlinked system and main protocols used were HTTP, HTML and URI[1]. It basically depended on HTTP pull model that a request can only be launched by users and then the server side can generate the contents and build pages using Server Side Includes(SSI) or Common Gateway Interface(CGI). Because of the less involvement and slower development of the client side, it's not hard to determine that the attack focal point at that period was the server side by manipulating HTTP requests.

2.2 Interpretation and discussion of OWASP TOP 10

Table 1. TOP Ten 2003[14]

A1 Unvalidated Parameters
A2 Broken Access Control
A3 Broken Account and Session Management
A4 Cross Site Scripting(XSS) Flaws
A5 Buffer Overflows
A6 Command Injection Flaws
A7 Error Handling Problems
A8 Insecure Use of Cryptography
A9 Remote Administration Flaws
A10 Web and Application Server Misconfiguration

As shown in the tables above, the most significant web security vulnerabilities in 2003 and 2004 overlap almost completely. It should be noted that these two top ten lists summarized by OWASP are figured out by surveying security researchers but without statistical measures, such as how frequently each vulnerability occurs, because in those days it was hard to find reliable resources and gather statistical information[13, 14]. Hence, they can be considered as two unordered lists.

Nonetheless, improper input validation looms above all. This category refers specifically to vulnerabilities that are triggered in the server-side and are prevalent in all web server environments. They are often generic, platform-independent or have little platform variability across server-side scripting environments. They

Table 2. TOP Ten 2004[13]

A1 Unvalidated Input
A2 Broken Access Control
A3 Broken Authentication and Session Management
A4 Cross Site Scripting(XSS) Flaws
A5 Buffer Overflows
A6 Injection Flaws
A7 Improper Error Handling
A8 Insecure Storage
A9 Denial of Service
A10 Insecure Configuration Management

exist mainly due to improper handling of HTTP requests where any part can be maliciously tampered by attackers. Unverified user input, when exploited, can lead to forced browsing, command execution, cookie poisoning and hidden field manipulation etc. There are two main reasons that caused an increasing number of these attacks. Firstly, input validation is only performed on the client side[13]. This can be a desirable feature for improving performance and usability for legitimate users. However, attackers can easily bypass the client-side defense mechanisms by using simple tools like telnet. Without support from server side, web applications have little resistance to malicious input. Secondly, input data filtering without proper "canonicalization" doesn't work as expected[13]. "Canonicalization" means that input should be converted to the simplest form before validation, otherwise vicious input can be masked with tricks and then survive in the filter. In fact, trying to sanitize user-supplied input by a white-list or black-list itself is difficult because there are multiple formats of encoding information which can be meanwhile easily decoded.

In the list above, (2003-A4 & 2004-A4) cross site scripting(XSS) flaws, (2003-A5 & 2004-A5)Buffer overflows and (2003-A6 & 2004-A6)Injection flaws are all concrete attack types due to improper input validation.

XSS The input used by XSS is malicious scripts usually in the form of embedded JavaScript that can be permanently stored in (Stored XSS) or reflected off (Reflected XSS) the web server. When such contents in a trusted server are requested and then retrieved, the victims' browsers will trustfully render the response and execute the code. Severe XSS vulnerabilities can be accessing to sensitive data kept by browser including cookies and session tokens, disclosure of end user files, modifying the HTML content and installing Trojan horse virus, etc. With a browser and available tools, hackers could find these bugs and craftily inject XSS attacks without much effort, so it showed high probability that a site was at the risk of XSS.

Buffer Overflow The buffer overflows refers to the overwritten program execution stack caused by overly large inputs. By sending carefully groomed input, attackers can take any over web application server component. For

example, if a graphics library is used to generate images, attackers can try to take control of the machine and let it run arbitrary image generating process until crashed. In fact, this kind of flaw was found commonly in widely used server products and meanwhile not easy to discover. Input size checking is always necessary for mitigating the damage and also helpful for catching other operational problems like (2004-A9)Denial of Service(DoS).

Injection Flaws Injection Flaws investigates tampered user input consisting of executable commands. By embedding malicious command in parameters and further passing them for accessing backend databases, operating system or other external systems, attackers want those commands to be executed. Whenever a web application tries to access an external interpreter, there is a potential risk of injection attack. So for some system calls and shell commands, instead of directly using operating system shell interpreter but using language-dependent libraries is a wise idea. And for necessary calls to backend databases, make sure that all supplied input that is finally treated as executable content goes through with careful test, which can reduce the risk to a great extent.

In general, in Web 1.0 measures of hacker's attack primarily targeted for tampering user input and in those days people still not had a strong sense of web security as very few valuable data were retained by others because of the read-only feature.

3 WEB 2.0 (OWASP TOP 10 in year 2007, 2010, 2013 and 2017)

3.1 Web 2.0: read-write interactive web

The term "Web 2.0" was first introduced by Darcy DiNucci in 1999 and has been widely spreaded by Tim O'Reilly and Dale Dougherty since 2004[3]. They proposed that internet should work as a platform where individual information can be exchanged and thus collective intelligence can be shared[2]. Users are not only allowed to read information, but are also encouraged to interact and collaborate with others and make their own contributions. Collectively classifying data, responsive web design, flexible information flows, RSS for subscribing and rapidly increasing user base can all be signified as huge successes of this participative social web[2]. Typical Web 2.0 tools includes blogs, podcasts, social networking e-voting and e-commerce. Particularly in technology, with the boom of agile development and dynamic scripting languages, Ajax, DOM, REST, many web front-end and back-end frameworks have emerged. They have become a very popular means of creating web contents, but brought inescapable security challenges at the same time. The trade-off between usability and security is still worth exploring. Compared with attack focusing on backend in Web 1.0, the front-end security of web applications has started to attract widespread attention in Web 2.0. Hackers are as well becoming more creative and skillful in exploiting different web technologies, which causes a variety of vulnerabilities.

3.2 Interpretation and discussion of OWASP TOP 10

Table 3. TOP Ten 2007[12]

A1 Cross Site Scripting(XSS)
A2 Injection Flaws
A3 Malicious File Execution
A4 Insecure Direct Object References
A5 Cross Site Request Forgery(CSRF)
A6 Information Leakage and Improper Error Handeling
A7 Broken Authentication and Session Management
A8 Insecure Cryptographic Storage
A9 Insecure Communications
A10 Failure to Restrict URL Access

Table 4. TOP Ten 2010[11]

A1 Injection
A2 Cross Site Scripting(XSS)
A3 Broken Authentication and Session Management
A4 Insecure Direct Object References
A5 Cross Site Request Forgery(CSRF)
A6 Security Misconfiguration
A7 Insecure Cryptographic Storage
A8 Failure to Restrict URL Access
A9 Insufficient Transport Layer Protection
A10 Unvalidated Redirects and Forwards

These four ordered lists has taken over a long span of time, so in total quite a few risk types made the lists, certainly several of them were renamed or grouped into a broadened category in each update. The technological prosperity in Web 2.0 has also brought considerable security threats. Among them, (2007-A1, 2010-A2, 2013-A3&2017-A7)cross site scripting(XSS) and (2007-A2, 2010-A1, 2013-A1&2017-A1)injection are even more remarkable.

Why is web so vulnerable to XSS in Web 2.0?

If you don't treat user supplied input properly before adding it to output pages, then unwanted scripts will be executed in the victims' browsers — that's what we have already noticed in Web 1.0. Different ways for generating output pages and different browser side interpreters make it quite difficult to realize effective

Table 5. TOP Ten 2013[10]

A1 Injection
A2 Broken Authentication and Session Management
A3 Cross-Site-Scripting(XSS)
A4 Insecure Direct Object References
A5 Security Misconfiguration
A6 Sensitive Data Exposure
A7 Missing Function Level Access Control
A8 Cross Site Request Forgery(CSRF)
A9 Using Components with Known Vulnerabilities
A10 Unvalidated Redirects and Forwards

Table 6. TOP Ten 2017[9]

A1 Injection
A2 Broken Authentication
A3 Sensitive Data Exposure
A4 XML External Entities
A5 Broken Access Control
A6 Security Misconfiguration
A7 Cross-Site-Scripting(XSS)
A8 Insecure Deserialization
A9 Using Components with Known Vulnerabilities
A10 Insufficient Logging & Monitoring

automative detection, and manual code review and penetration testing are obligated. Technologies in Web 2.0, for instance, Ajax encompassed by JavaScripts, has no inherent security drawbacks but could expose more security vulnerabilities in an insecure design. A Web 2.0 application consists of lots of Ajax calls for dynamically updating the pages, which makes automated detection more difficult due to the invisibility of these calls. It's easy to arouse validation confusion as well when the third party such as Mashups or Widgets is used. Believing that others implemented the validation so mine is unnecessary leads to none of them effectively making validation control.

Moreover, DOM-based XSS matters. Unlike the traditional XSS attack, such as stored XSS and reflected XSS, the problem comes from performing data processing on the server side. In DOM-based XSS, the malicious data does not touch the server but is inserted into the source code and then parsed as HTML. This is even worse because any user input that does not reach the server side can be dangerous to the users. Modern web applications have a tendency to be single page and several JavaScript frameworks are successful in completing such an application, the direct result is that an increasing amount of HTML is generated on the client side rather than on the server side, which also increases the risk of DOM-based XSS.

What is also worthy of remark is XSS worms, which attacked millions of users on the social networking sites[4]. When users visit a page, the self propagating XSS code will spread quickly. Its exponential growth has provoked widespread public debate on security and privacy concerns.

Why is web so vulnerable to injection in Web 2.0?

Similar to XSS, "trust without enough validation" is the root cause of injection flaws. In Web 2.0, injection attacks, especially SQL injection is quite prevalent. Large sets of critical data are stored in databases, which are extremely attractive to hackers. Reliable database management can be considered as a core competitiveness of web 2.0 companies. On that stage, SQL injection was also used to exploit other web 2.0 technologies, which makes the situation worse. By means of SQL injection, hostile swf files and malware serving JavaScript can be injected into the application. There are many human made carelessness that may result in insecure database access. Making use of outdated or unpatched software may expose security flaws with modern software, for example, the prevalence of older functional interfaces caused many PHP and ASP applications under SQL injection attack. Therefore, running patched version and paying timely attention to new vulnerabilities are important for preventing security flaws.

Feed injection, is a specific Web 2.0 injection vulnerability and can be used to escalate to more serious attack such as XSS and SQL injection[5]. "Subscription" is a typical feature of Web 2.0, which is realized by utilizing XML content

feeds following RSS and Atom standards. This gives both users and websites the opportunity to extract some information including headline and summary of the body without entering the sites. RSS provides flexibility and availability to push information to the end user, but the involved security risk is high. Free accessibility for end-users, absolute control over the feeds consume and uncertain third party sources all lead to insecure RSS and can be exploited by hackers.

More web attack on client side

In Web 1.0 hackers focused on attacking server side, while in Web 2.0 they also pay attention to client side vulnerabilities. This trend is unstoppable. Compared with server side attack, there's no need to install any software, attackers can execute payloads and retrieve information via less overhead cost. Apart from DOM-based XSS and feed injection, (2007-A5, 2010-A5 and 2013-A8)Cross Site Request Forgery(CSRF) is another common attack on client side. CSRF exploits the feature that browsers send credentials automatically, so it is hard to distinguish forged requests generated by numerous tricks from the legitimatize ones. If the victim is authenticated and then unintentionally push the forged request, the CSRF attack happens. Fortunately, it is easy to detect this vulnerability with the help of code review and penetration testing. As we can see from the Top 10 list in 2017, it no longer exists as a severe vulnerability.

4 WEB 3.0 (OWASP TOP 10 in year 2021) and future web

4.1 Web 3.0: read-write intelligent web

Different Web experts once expressed themselves to Web 3.0. "A semantic Web" where all the data on the Web can be read and handled by machines was introduced by Tim Berners-Lee. Take advantage of metadata and with the support of world wide databases, the visual information can be more meaningful. More practically, in Yahoo founder, Jerry yang' s opinion, the gap of creating a program between professionals and amateurs should be narrowed or blurred with the help of developed tools and techniques[1], which leads to another key feature of Web 3.0 — decentralization. Of interest, Follow, the first decentralized social protocol for Web 3.0 based on block chain architecture, was officially released on Sep. 22, 2021 and it allows users have full control over their own data. In Web 2.0, the most of the internet's critical services belonged to few giant companies, so it's desirable for users to own better self control. However, it also comes with problems when users dominate. In the context of a centralized Web, governments can make large companies enforce local policies and laws, while it is difficult and unclear to conduct in a decentralized Web if the web contents created by individuals are hosted all over the world and the digital identity may be not consistent with the real identity.

The intensive cooperation of public and private data make Web 2.0 highly attractive to web users and hackers as well. The same case also occurs in Web 3.0. On account of new technologies for artificial intelligence and big data, users are exposed to more severe, more immediate information leakage. The answer to "Who owns the data and who can use the data" is still obscure and controversial since the concept "internet as a platform" for Web 2.0 has been put forward.

4.2 Interpretation and discussion of OWASP TOP 10

Table 7. TOP Ten 2021[8]

A1 Broken Access Control
A2 Cryptographic Failures
A3 Injection
A4 Insecure Design
A5 Security Misconfiguration
A6 Vulnerable and Outdated Components
A7 Identification and Authentication Failures
A8 Software and Data Integrity Failures
A9 Security Logging and Monitoring Failures
A10 Server-Side Request Forgery(SSRF)

Compared with the publishing from five years ago, there are several changes in recent update of OWASP Top 10 from last year. They bears thinking about in order to have a better insight about security issues in Web 3.0 and future Web.

Broken Access Control Although this flaw has not been discussed much above, it has always been on the list and moves to the first place in the recent update, or to say, access control is a timeless topic in the field of web security and other vulnerabilities are more or less related to or caused by wrong implementation of access control. It discusses the problem that users can access some information or perform some actions they are not allowed to do. The front-end and back-end decoupling in modern web applications gives hackers the chance to exploit access control to impersonate legitimate users. Code exploits are inevitable, so prompt diagnosing weak points by penetration testing is helpful for mitigating this problem. In the future Web, more business that contains more sensitive data will be conducted online. If not take key consideration of access control, web applications will wide open to problems such as data theft and privacy leakage.

Cryptographic Failures This was previously named as Sensitive Data Exposure[9] and shifts up one position to the second place. It is concentrated on the vulnerabilities related to cryptography. The severity of this problem in

the future depends on the development of encryption algorithms, the processing speed of machines and how cryptography will be regulated to protect sensitive data in the web applications, which needs the guidance from corresponding privacy laws. In addition, cryptocurrency — a decentralized digital currency made of cryptographic techniques, is also a heated topic in Web 3.0. Decentralization means that anyone can sell or buy cryptocurrency without verification by banks in traditional trade, so this vulnerability may require more focus in the future.

Injection In the new release, XSS is merged to injection. This vulnerability has still remained a high-ranking position of the OWASP Top 10 year after year, even though it is known and has been studied for over two decades. Nowadays the market is full of commercial and open-source tools which claims they are sure of eliminating them, but it's interesting that even large organizations fail to overcome injection, such as SQL injection, very frequently. The root cause does not change. Developers are used to trusting that the source of database queries is reliable, so they are lack of security awareness of validating user input and protecting sensitive data. Database access API should offer a safe way for retrieving data, but it is often not well structured by developers, for example, the imprecise use of prepared statements. In future Web, end-users will still be confronted with serious data breaches caused by database injection. Almost all web applications rely on back-end databases for information storage and retrieval. The huge quantity of data sets itself provides a certain space for attack. And currently almost no automated and efficient methods that can test out SQL injections very precisely ("precise" means no false positives)

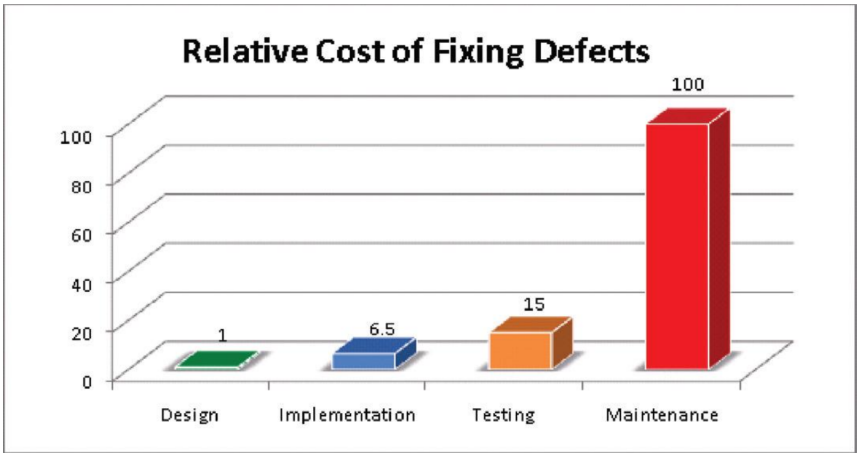

Fig. 2. Relative Cost of Fixing Defects [6]

Insecure design This is a totally new category. It's limited or missing in current web development, so putting it here is instructive for future web development. It discusses the security risks to do with design and architectural vulnerabilities and appeals for more secure development life cycle, more secure design pattern and more use of threat modeling. Insecure design and insecure implementation are distinct concepts. An application with secure design can still be not immune to attack because of implementation defects, but a highly secure implementation pattern can definitely not remedy an insecure design. From the graph above, the relative cost for fixing a problem is the lowest. Therefore, it is worthwhile to pay more attention in the design stage.

Security Misconfiguration Security misconfiguration has been a problem since the Web was born. Having a strong configuration standard is important to secure a web application. With the popular shift from "customization" to "configuration" in the software world, it is predictable that the misconfiguration will still be a moving up problem in the future.

Identification and Authentication Failures This was previously named as Broken Authentication and moves down to the eighth place. The increased availability of standardized authentication frameworks that adopts multi factor authentication(MFA) has a positive impact. Now included serious CWEs are more related to identification failures, such as (CWE-620)unverified Password Change and (CWE-640) Weak Password Recovery Mechanism For Forgotten Password[8]. In a decentralized Web, the users will have more control over their own identity rather than centralized management by large influential organizations, there may be more attack on extracting users's identity data, so it's necessary to equip the web applications with layer-to-layer security check.

5 As a Web Developer

Chris Wysopal said: "Web developers need to know that the degree to which business applications and customer data are protected from the hostile internet is directly determined by how securely they've written their code[13]." Exactly, to some extent, the vulnerability prevention, such as avoid using outdated components previously mentioned, is the responsibility of people, namely web developers. A few suggestions that works not limited to web development as follows:

For beginners, clearly understand the exact cause and location of attacks. In the field of Web security, the industry is used to dividing it into two parts: client-side (front-end) security and server-side(back-end) security, because such a division is widely accepted in web application development. It does not take into account the more specific differences in the conditions and locations of how a vulnerability is triggered. Such a division is not conducive for systematic learning and in-depth understanding of web application security principles for beginners.

Raise security awareness from the basics when programming. Developers may hold the opinion that "secure" is a more high-level requirement and thus it is quite complex to realize. Not really, in fact sometimes only subtle improvement in coding habits can secure the application a lot. For example, always use a hash for data exchange to ensure the data integrity. OWASP provides negative examples for each vulnerability due to insecure programming, which can be used as reference.

Use tools wisely. At a time when frameworks but not languages rule the world of the Web, developers can worry less about security by making use of those frameworks. However, relying on those tools alone results in that developers lose security awareness and implicitly trust in that frameworks already build enough security modules, but in fact they are not completely reliable. For example, v-html template from Vue.js allows raw HTML rendering without data filtering, and its documentation already states that inadvisable use of this template might expose users to XSS attack. Beyond that, JavaScript itself has some functions, mostly for script execution or DOM manipulation, which should be used with caution.

6 Conclusion

To conclude, in the evolution of Web from 1.0 to 3.0, users enjoy the convenience of flourishing web technologies, but also faces security concerns, which is inevitable. In Web 1.0, the focal point of security issues was on the server side and tampering the user input is the most prevalent way to attack server. Then a Web 2.0 application allowed collaboration from users and interaction between webpages and users became more diversified, which also made client side attractive to hackers. In Web 3.0 and future Web, users expect an intelligent and decentralized web where they can take full control of their data but also feel anxious about data breaches and privacy leakage.

References

1. Nath, K., Dhar, S., Basishtha, S.: Web 1.0 to Web 3.0 - Evolution of the Web and its various challenges. Conference: 2014 International Conference on Optimization, Reliabilty, and Information Technology (ICROIT)(2014) https://doi.org/10.10007/123456789010.1109/ICROIT.2014.6798297
2. Hiremath, B., Kenchakkanavar, A.Y.: An Alteration of the Web 1.0, Web 2.0 and Web 3.0: A Comparative Study(2016)
3. Aslam, S., Sonkar, S.K.: Journey of Web 2.0 to Web 3.0. Conference: Empowering Libraries with Emerging Technologies for Common Sustainable FutureAt: Babasaheb Bhimrao Ambedkar University, Lucknow. (2019)
4. Chaudhary, P., Gupta, B.B., Gupta, S.: Cross-Site Scripting (XSS) Worms in Online Social Network (OSN): Taxonomy and Defensive Mechanisms. Conference: Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on, India(2016)

5. Auger, R(SPI Labs).: Feed Injection in Web 2.0 - Hacking RSS and Atom Feed Implementations, pp.3(2006)
6. Dawson, M., Burrell, D.N., Rahim, E., Brewster, S. :Integrating Software Assurance into the Software Development Life Cycle (SDLC). JISTP-Volume 3, Issue 6(2010), pp.49–53 (2010)
7. Benhaddi, M.: Web of goals: A proposal for a new highly smart web, pp. 2. Proceedings of the 19th International Conference on Enterprise Information Systems (2017). https://doi.org/10.5220/0006250306870694
8. OWASP TOP 10 2021, https://owasp.org/www-project-top-ten/. Last accessed 14 Feb 2022
9. OWASP TOP 10 2017, https://owasp.org/www-project-top-ten/2017/. Last accessed 14 Feb 2022
10. OWASP TOP 10 2013, https://owasp.org/www-pdf-archive//OWASP_Top_10_-_2017_Release_Candidate1_English.pdf. Last accessed 14 Feb 2022
11. OWASP TOP 10 2010, https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2010.pdf. Last accessed 14 Feb 2022
12. OWASP TOP 10 2007, https://owasp.org/www-chapter-belgium/assets/2007/2007-03-21/OWASP%20Intro%20and%20Top%2010%202007.pdf. Last accessed 14 Feb 2022
13. Weaknesses in OWASP TOP 10 2004, https://cwe.mitre.org/data/definitions/711.html. Last accessed 14 Feb 2022
14. OWASP TOP 10 2003, https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2013.pdf. Last accessed 14 Feb 2022