International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.1211
(09/2014)

SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

Cybersecurity security – Cybersecurity

Techniques for preventing web-based attacks

Recommendation ITU-T X.1211

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	X.1–X.199
OPEN SYSTEMS INTERCONNECTION	X.200–X.299
INTERWORKING BETWEEN NETWORKS	X.300–X.399
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	X.600–X.699
OSI MANAGEMENT	X.700–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	X.850–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
INFORMATION AND NETWORK SECURITY	
General security aspects	X.1000–X.1029
Network security	X.1030–X.1049
Security management	X.1050–X.1069
Telebiometrics	X.1080–X.1099
SECURE APPLICATIONS AND SERVICES	
Multicast security	X.1100–X.1109
Home network security	X.1110–X.1119
Mobile security	X.1120–X.1139
Web security	X.1140–X.1149
Security protocols	X.1150–X.1159
Peer-to-peer security	X.1160–X.1169
Networked ID security	X.1170–X.1179
IPTV security	X.1180–X.1199
CYBERSPACE SECURITY	
Cybersecurity	**X.1200–X.1229**
Countering spam	X.1230–X.1249
Identity management	X.1250–X.1279
SECURE APPLICATIONS AND SERVICES	
Emergency communications	X.1300–X.1309
Ubiquitous sensor network security	X.1310–X.1339
CYBERSECURITY INFORMATION EXCHANGE	
Overview of cybersecurity	X.1500–X.1519
Vulnerability/state exchange	X.1520–X.1539
Event/incident/heuristics exchange	X.1540–X.1549
Exchange of policies	X.1550–X.1559
Heuristics and information request	X.1560–X.1569
Identification and discovery	X.1570–X.1579
Assured exchange	X.1580–X.1589
CLOUD COMPUTING SECURITY	
Overview of cloud computing security	X.1600–X.1601
Cloud computing security design	X.1602–X.1639
Cloud computing security best practices and guidelines	X.1640–X.1659
Cloud computing security implementation	X.1660–X.1679
Other cloud computing security	X.1680–X.1699

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T X.1211

Techniques for preventing web-based attacks

Summary

Recommendation ITU-T X.1211 describes techniques that can mitigate web-based attacks which occur when the vulnerabilities of the website hosts are exploited and malicious code is introduced that can infect a user's computer. Several appendices illustrate how the attacks can occur as well as remediation steps.

History

Edition	Recommendation	Approval	Study Group	Unique ID[*]
1.0	ITU-T X.1211	2014-09-26	17	11.1002/1000/12154

Keywords

Prevention, SQL injection, spyware, suspicious content, vulnerability, web-based attack.

[*] To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

Table of Contents

Page

1 Scope ... 1

2 References ... 1

3 Terms and definitions ... 1

 3.1 Terms defined elsewhere ... 1

 3.2 Terms defined in this Recommendation ... 2

4 Abbreviations and acronyms ... 2

5 Conventions ... 3

6 General overview ... 3

7 Web-based attack protection system techniques ... 4

 7.1 General techniques ... 4

 7.2 Functional techniques ... 4

 7.3 Management techniques ... 5

 7.4 Security and privacy techniques ... 5

8 Functions of the web-based attack protection system ... 5

9 Information exchange format ... 6

Appendix I – Scenarios for web-based attacks ... 7

 I.1 Scenario for malware infection ... 7

 I.2 Cross-site request forgery (CAPEC-62) ... 7

 I.3 Cross-site port attacks/server-side request forgery ... 8

 I.4 SQL injection ... 8

 I.5 Detecting malware in websites ... 9

Appendix II – Method for infecting user computers with malware ... 10

Appendix III – Typical examples of obfuscation technique ... 11

Appendix IV – Prevention techniques for web-based attacks ... 12

 IV.1 Remove website vulnerabilities ... 12

 IV.2 Signature matching ... 12

 IV.3 Site blacklisting ... 12

 IV.4 Detection of obfuscating techniques ... 12

 IV.5 Evaluation of suspicious content behaviour ... 12

Appendix V – Typical examples of application security risks by OWASP ... 13

Bibliography ... 22

Recommendation ITU-T X.1211

Techniques for preventing web-based attacks

1 Scope

This Recommendation provides techniques for preventing web-based attacks. It describes the use scenarios to distributing malwares through the web as well as the functional techniques and functions to prevent web-based attacks.

2 References

None.

3 Terms and definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 asset [b-ISO/IEC 27000]: Anything that has value to the organization.

NOTE – There are many types of assets, including:

a) information;

b) software, such as a computer program;

c) physical, such as computer;

d) services;

e) people, and their qualifications, skills, and experience; and

f) intangibles, such as reputation and image.

3.1.2 attack instance [b-ITU-T X.1544]: A specific detailed attack against an application or system targeting vulnerabilities or weaknesses in that system.

3.1.3 attack pattern [b-ITU-T X.1544]: An abstraction of common approaches of attack observed in the wild against applications or systems (e.g., SQL injection, man-in-the-middle, session hijacking).

NOTE – A single attack pattern may potentially have many varying attack instances associable with it.

3.1.4 hypertext markup language (HTML) [b-ITU-T M.3030]: A system of coding information from a wide range of domains (e.g., text, graphics, database query results) for display by World Wide Web browsers. Certain special codes, called tags, are embedded in the document so that the browser can be told how to render the information.

3.1.5 malware [b-ISO/IEC 27033-1]: Malicious software designed specifically to damage or disrupt a system, attacking confidentiality, integrity and/or availability.

3.1.6 obfuscation technique [b-NIST SP 800-83]: A way of constructing a virus to make it more difficult to detect.

3.1.7 personally identifiable information (PII) [b-ITU-T X.1252]: Any information a) that identifies or can be used to identify, contact, or locate the person to whom such information pertains; b) from which identification or contact information of an individual person can be derived; or c) that is or can be linked to a natural person directly or indirectly.

3.1.8 threat [b-ITU-T X.800]: A potential violation of security.

3.1.9 security domain [b-ITU-T T.411]: The set of resources subject to a single security policy.

3.1.10 security domain authority [b-ITU-T X.810]: A security authority that is responsible for the implementation of a security policy for a security domain.

3.1.11 security policy [b-ITU-T T.411]: The set of rules that specify the procedures and services required to maintain the intended level of security of a set of resources.

3.1.12 signature [b-NIST SP 800-83]: A set of characteristics of known malware instances that can be used to identify known malware and some new variants of known malware.

3.1.13 spyware [b-NIST SP 800-83]: Malware intended to violate a user's privacy.

3.1.14 web browser plug-in [b-NIST SP 800-83]: A mechanism for displaying or executing certain types of content through a Web browser.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 anomaly: A pattern in the data that does not conform to the expected behaviour.

3.2.2 drive-by-download attacks: A pattern of a web-based attack caused when a user visits a website that exploits browser vulnerabilities and launches the automatic download and installation of malware without the knowledge or permission of the user.

3.2.3 web-based attack: A pattern of attacks in which the attackers compromise the legitimate websites resulting in a malicious code to be injected into an application, which in turn can be used to infect the user's computer visiting those websites or use vulnerabilities of web sites to launch attacks for user's computer systems that visit that web sites, which occurs without involvement of malware.

3.2.4 web-based attack protection system: A set of systems which detects vulnerabilities, malwares or malicious codes embedded in the legitimate website and informs the web administrator of the detection result, leading ultimately to their removal.

NOTE – Detection activities may be planned by schedule or may be triggered by network events or requests from other systems.

3.2.5 zombie computer: A computer that has been compromised and controlled by an attacker who has installed malwares such as computer viruses, Trojan horse, or bot net, which can be used to perform malicious attacks such as spreading e-mail spams and launching denial-of-service attacks.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

CAPEC	Common Attack Pattern Enumeration and Classification
CSRF	Cross-Site Request Forgery
CWE	Common Weakness Enumeration
DDoS	Distributed Denial of Service
DOM	Document Object Model
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identity
IODEF	Incident Object Description Exchange Format
LDAP	Lightweight Directory Access Protocol

MITM	Man-in-the-Middle
OS	Operating System
OWASP	Open Web Applications Security Project
PC	Personal Computer
PII	Personally Identifiable Information
PUI	Program Under Inspection
SNS	Social Network Service
SQL	Structured Query Language
SSRF	Server-Side Request Forgery
S/W	Software
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XSPA	Cross-Site Port Attack
XSS	Cross-Site Scripting

5 Conventions

None.

6 General overview

Malware that is used to comprome information assets is defined as software designed specifically to damage or to disrupt a system, attacking confidentiality, integrity and/or availability. It includes computer viruses, worms, Trojan horses, spyware, adware, most rootkits and other malicious programs.

A web-based attack is an attack whereby the attackers try to compromise the legitimate websites by exploiting existing vulnerabilities. This results in malicious code to be injected into the websites, which can in turn be used to infect the computers of users visiting those websites. The malicious code may have multiple forms: it can be a hidden iframe tag directing the user to visit an attack site, or it can be malicious applications written in a computer program language (e.g., script or applets). Typical examples of vulnerabilities of web-based attacks are Structured Query Language (SQL) injection, and cross-site request forgery (CSRF).

A cross-site request forgery attack pattern [b-CAPEC-62] is a type of web-based attack whereby unauthorized commands are transmitted or unwanted actions are requested to be executed on a trusted website without the user's knowledge while the user is logged into a trusted website. A Structured Query Language (SQL) injection attack pattern [b-CAPEC-66] is another type of web-based attack on a database-driven website in which the attacker adds an SQL code to a web from an input box to gain access to resources or make changes to data. It is used to steal information from a database from which the data would normally not be available and/or to gain access to an organization's host computers through the computer that is hosting the database. An in-line frame, also known as iframe tag [b-iframe], is used to embed an invisible document within the current hypertext markup language (HTML) document and tricking the user to click on the invisible document through clickjacking [b-CAPEC-103].

Recently, web-based attacks have been increasing significantly due to increasing use of end-user computing devices and the increasing number of websites that contain malware.

Anti-virus techniques could be implemented at the server side and web application firewalls could be implemented at proxies for cost-effective implementation of these techniques.

In web-based attacks, the administrators of the websites may not be aware that the websites have been hacked and injected with malicious code, and that these are used to disseminate malicious code. Moreover, users are not aware either that their computers may get infected by malicious code from the sites they have visited. Installing anti-virus software (S/W) can prevent some incidents, but does not provide ultimate solutions.

The reasons for an increase in web-based attacks are as follows:

- Drive-by-download attacks from mainstream web sites are increasing;
- Attacks are heavily obfuscated and dynamically changing, making traditional malware detection and prevention solutions ineffective;
- Attacks are targeting web browser plug-ins of end users;
- SQL injection attacks are being used to infect mainstream websites;
- Malicious advertisements are redirecting users to malicious websites; and
- An explosive growth in unique and targeted malware samples.

7 Web-based attack protection system techniques

7.1 General techniques

The following techniques are characteristic of the web-based attack protection system:

- designed to be scalable, robust and resilient;
- operated across multiple security domains, each of which is managed by a responsible security administrator; and
- exchange information about website vulnerabilities or malware-infected websites (i.e., websites with invisible iframe redirecting users to the malware-infected website [b-CAPEC-103]);

NOTE – The existing incident object description exchange format (IODEF) [b-ITU-T X.1541] could be used to exchange information.

- operated in one of the two types of deployment models: a centralized model and a distributed model. In the centralized model, all information about the malware-infected websites and types of malware should be reported to, maintained or controlled by the centralized server. In the distributed model, each security domain should set up a responsible agent and information about the malware-infected websites and types of malware should be exchanged among the responsible agents that exist in distributed locations;
- configured in a hierarchical manner to facilitate the scalable operation.

7.2 Functional techniques

The following functional techniques are characteristic of the web-based attack protection system:

- identify known malware from legitimate web content and prevent websites from the malware being installed;
- detect the invisible iframe that redirects the user to other websites that install malware;
- detect vulnerabilities that can be used for typical web-based attacks such as SQL injection, cross-site reference, etc., as described in Appendix IV;
- conduct signature-based analysis or equivalent analysis to detect the known malware in the website;

- conduct behaviour-based analysis for identifying unknown malware;
- inform the administrator of the website of malware infection to remove malware in the websites;
- detect obfuscated malware using string splitting, string encoding, custom string encoding, script behaviour modification, obfuscating document object model (DOM) modification functions, hiding links behind public services and page redirections in the website;
- detect malware which can be used for cross-site reference forgery attacks in the websites;
- evaluate behaviours of suspicious malware in the websites;
- inform users about infected websites in case a user visits those infected websites;
- when a web-based attack protection system detects malware in a website, inform the security administrator that the website has been infected with malicious code and that it can ultimately be used for a web-based attack;
- exchange information about blacklists of malicious websites; and
- identify website vulnerabilities including SQL injection and cross-site scripting, and inform the administrator of those websites of their identified vulnerabilities.

7.3 Management techniques

The following management techniques are characteristic of the web-based attack protection system:

- support security management based on security policies when being deployed in different security domains;
- have a unified interface to support management for a centralized management system;
- support trust management and only accept attack-related event data from the trusted security domains;
- support the system resource management and protect the system from being overloaded; and
- support operation and maintenance management including system configuration management, log management, system status monitoring, etc.

7.4 Security and privacy techniques

The following security and privacy techniques are characteristic of the web-based attack protection system:

- provide confidentiality, data origin authentication and integrity of information exchanged through the communication interface between security domains;
- prevent leakage of personally identifiable information (PII) which the web-based prevention system processes;
- provide resilience to various network-based attacks, for example, distributed denial of service (DDoS) attacks; and
- provide an auditing functionality which can trace misuse or abuse of information collected for the web-based attack protection system by unauthorized entities.

8 Functions of the web-based attack protection system

The web-based attack protection system should provide at least, but is not limited to, the following functions:

- Detection of all known vulnerabilities in the websites;
- Detection of websites that contain malware used for malware distribution;

- Notification of the administrator of websites that contain malware and have known vulnerabilities that can be exploited by attackers;
- Collection of the necessary information about the vulnerabilities of the websites and the malware that they contain;
- Sharing of information about malware-infected websites and those that are used for malware distribution between trusted entities in a security domain and among multiple domains;
- Implementation of the security policy of the web-based protection system in a domain; and
- Protection of the web-based attack protection system from any attacks.

9 Information exchange format

Information sharing about exchanging malware analysis information (e.g., malware attribute enumeration and characterization) should be reinforced. The implementers of this Recommendation may use [b-ITU-T X.1546] for exchanging malware analysis information.

Appendix I

Scenarios for web-based attacks

(This appendix does not form an integral part of this Recommendation.)

I.1 Scenario for malware infection

Figure I.1 depicts a typical scenario of web-based attacks.

1. Attackers compromise a legitimate website which has vulnerabilities and then install a malware or a script which is used to attack the user's computer or install tags to redirect the user's access to the website that contains the malware to attack the user's computer which has visited that website.
2. When a user, a victim, visits the website which has been compromised by the attackers, the user's computer is attacked by the malware embedded or is redirected to another website which contains malware to attack the user's computer.
3. When the user's computer has browser vulnerabilities which can be used by the specific malware, the user's computer is infected by that malware without the knowledge or permission of the user.
4. The malware installed in the user's computer might be used to launch massive distributed denial of service (DDoS) attacks or to steal personal information such as identity (ID) and password which are then forwarded to the attackers.

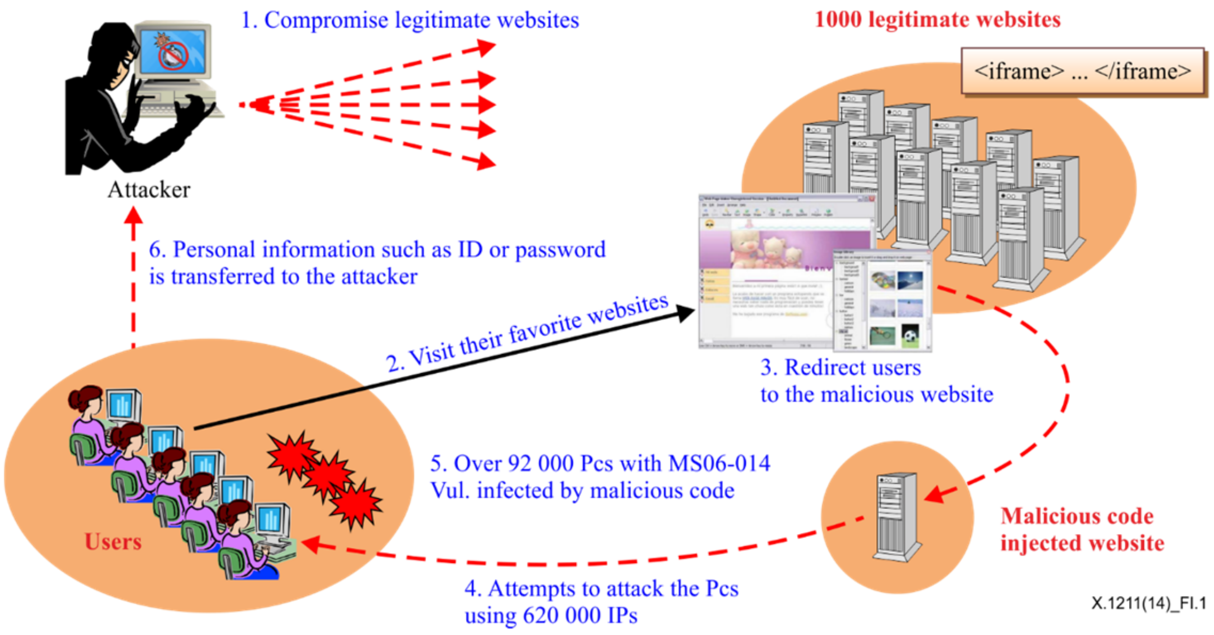
1. Compromise legitimate websites
1000 legitimate websites
<iframe> ... </iframe>
Attacker
6. Personal information such as ID or password is transferred to the attacker
2. Visit their favorite websites
3. Redirect users to the malicious website
Users
5. Over 92 000 Pcs with MS06-014 Vul. infected by malicious code
Malicious code injected website
4. Attempts to attack the Pcs using 620 000 IPs
X.1211(14)_FI.1

Figure I.1 – Typical scenario of web-based attacks

I.2 Cross-site request forgery (CAPEC-62)

The cross-site request forgery (CSRF) may cause a victim to unwittingly submit one or more hypertext transfer protocol (HTTP) requests to a vulnerable website that a user trusts. A typical cross-site request forgery attack may compromise data integrity accordingly and give an attacker the ability to modify information stored by a vulnerable website.

When a website requires user authentication, it often does not require a user to type in their password for every HTTP request. Instead, a website identifies a user's authentication state between multiple HTTP requests by tokens such as session cookies or the HTTP authorization header. However, there is a problem: web browsers memorize the token associated with a uniform resource locator (URL) and automatically attach the token when a new HTTP request is issued to the website, even if the request is not intended by the user. CSRF takes advantage of the browser's behaviour. With CSRF, a user just needs to visit a malicious website that can include JavaScript logic that issues (potentially hidden) HTTP requests to other websites (such as the user's bank), and those HTTP requests might be authorized by the website because of the presence of the tokens. CSRF enables various kinds of various attacks, such as sending e-mails from a web-based mail service, posting a comment to a blog on the user's behalf, altering the user's buddy list in social network service (SNS) or changing settings in a home router.

I.3 Cross-site port attacks/server-side request forgery

Cross-site port attacks/server-side request forgery (XSPA/SSRF) is a method of abusing web-applications that process URLs provided by a web-browser input. A typical XSPA/SSRF attack is targeted at the intranet of the vulnerable application. The attack may cause port scanning, compromise data confidentiality, lead to unauthorized code execution and exploitation of vulnerable intranet resources. The application is considered vulnerable to XSPA/SSRF when it does not validate the output received from a remote host and the input provided by the end user. As an example, the application that downloads an image from a URL provided by a user could access an intranet resource when the user posts the URL, as 'http://localhost/secret.txt'. In some cases, special uniform resource identifier (URI) schemas may be used so that a vulnerable application would send a request to special services such as 'https', 'gopher', 'ftp' or 'ldap'. Language-specific schemas such as 'php://fd', 'php://memory' could be used as well.

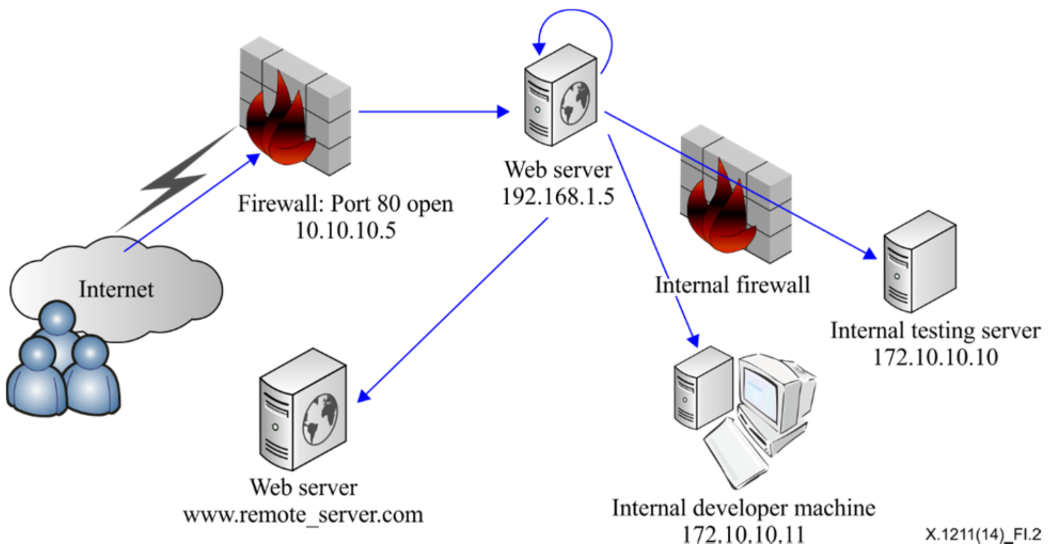

Figure I.2 – Typical scenario of cross-site port attacks/server-side request forgery

I.4 SQL injection

A typical SQL injection [b- CAPEC-66] scenario is based on a poor sanity check of input data for web-applications. Input channels may vary from GET and POST HTTP-requests, browser cookies, XML-based payloads, file inputs and others.

Targeted input is then injected into the SQL-query. Here is the basic example of SQL-injection in a HTTP "GET" parameter:

- Given the original query – "SELECT title, content FROM table1 WHERE id = %d" where, "id" is the targeted parameter.

Under normal conditions "id" is some natural number. But, due to lack of sanity check instead of a number, the attacker may provide the following input:

- %d = "1 UNION SELECT user, password FROM secret_table".

This would lead to unauthorized access to the "secret_table" resulting in the disclosure of sensitive data directly in the browser's output.

Depending on the SQL-database implementation such attack could lead to:

- disclosure of sensitive data from database or file system;
- data loss/modification;
- injection of backdoors and privileges escalation; and
- deployment of malware to end-users visiting the site.

I.5 Detecting malware in websites

Techniques used for detecting malware can be grouped into two categories: anomaly-based detection and signature-based detection [b-NA].

In an anomaly-based detection technique, the criteria for determining maliciousness of a program under inspection are what constitute normal behaviours. A special type of anomaly-based detection is referred to as specification-based detection. Specification-based detection techniques use some specification or rule set of valid behaviour in order to decide the maliciousness of a program under inspection. Programs violating the rule set or specification are regarded as malicious.

In a signature-based detection, the criteria for determining the maliciousness of a program under inspection are the characterization of what is known to be malicious. The characterization or signature of malicious behaviour is the key to a signature-based detection method's effectiveness.

Each of the detection techniques can employ one of three different approaches: static, dynamic, or hybrid. The specific approach or analysis of an anomaly-based or signature-based technique is determined by how the technique gathers information to detect malware. Static analysis uses syntax or structural properties of the program (static)/process (dynamic) under inspection (PUI) to determine its maliciousness. For example, a static approach to signature-based detection would only use structural information (e.g., sequence of bytes) to determine the maliciousness, whereas a dynamic approach will use runtime information (e.g., systems seen on the runtime stack) of PUI.

In general, a static approach attempts to detect malware before the program under inspection executes. Conversely, a dynamic approach attempts to detect malicious behaviour during program execution or after program execution.

There are hybrid techniques that combine the two approaches. In this case, static and dynamic information is used to detect malware.

There are several detection techniques for malware in websites; these are described in Appendix III.

Appendix II

Method for infecting user computers with malware

(This appendix does not form an integral part of this Recommendation.)

This appendix describes typical scenarios that could be used by attackers, in order to help administrators understand them.

The first step for a web-based attack is to install and run various malicious codes on a user's computer. The malicious codes may include keystroke loggers and rootkits (which can turn user computers into zombie computers or leak sensitive user information to attackers).

The objective of the attack could be achieved by either exploring several known vulnerabilities of different software components accessible from a browser (e.g., operating system components accessible from a browser through ActiveX, etc.), or through attack techniques using social engineering to trick users into installing and running malware on their system. In addition, this attack attempts to steal user's credentials by phishing techniques or cross-site scripting attacks run in a hidden iframe.

There are a number of techniques which are used to infect a user's computer with malware: exploiting an ActiveX component, social engineering techniques, missing codec, malware removal tool techniques and cross-site request forgery attacks. Detailed information can be found in [b-NTobjectives]. In addition, there is a list of common attack patterns along with a complete list of schema and classification in [b-ITU-T X.1544].

Appendix III

Typical examples of obfuscation technique

(This appendix does not form an integral part of this Recommendation.)

The injected malicious content uses an obfuscated technique in order to hide malware both from the human eye and vulnerability [b-ITU-T X.1520] detection software. Obfuscating techniques are quite effective due to the following reasons:

- Many website administrators are wary of deleting script codes they do not understand.
- Database administrators have trouble cleaning infected databases, not knowing which patterns to look for.
- Many detection methods rely on regular expression or other string search-related methods, and thus have problems identifying obfuscated HTML.

There are several obfuscation methods: string splitting, string encoding, custom string encoding, script behaviour modification, obfuscating DOM modification functions, hiding links behind public services and page redirection. Detailed information is described in [b-NTobjectives].

Appendix IV

Prevention techniques for web-based attacks

(This appendix does not form an integral part of this Recommendation.)

This appendix presents several techniques for detecting malware in websites [b-NTobjectives]. The malicious content can be detected by content signature matching, blacklisting attack sites or analysing content for suspicious behaviour by proprietary algorithms.

IV.1 Remove website vulnerabilities

The simplest way is to remove websites vulnerabilities, including SQL injection and cross-site scripting. If the attacker is not able to insert malicious content into the website, client browser will not execute the malware inserted in the website. Therefore, the most efficient way to prevent web-based attacks is to remove all vulnerabilities from websites.

IV.2 Signature matching

Since there are a number of obfuscation techniques and automation tools to obfuscate malware, it is impractical to detect malware content in the website by using a signature-based detection method. It is well known that attackers are able to automate encoding malicious content with a new key for each website, thus resulting in creating a different signature of malware for each website. However, plain malware content is not changed frequently, and thus malware in the website can be detected with a signature. If plain malware content is obtained by decoding the encoded malware and the signature of plain malware is calculated from plain malware, this method can detect the malware by comparing the calculated signature of malware with a precompiled list of all known malware content signatures .

IV.3 Site blacklisting

Blacklisting attack websites is among the most valuable detection techniques. Although malicious content can be completely hosted on a good website (with no requirements to load automatically any scripts or iframes from an attack site, thus hiding their connection to the attack site), it is necessary to exchange some data with the attack website to complete the intended attack. This necessary data exchange can have many different forms: the attack script needs to download malware from the attack website, or send gathered private data from the users' system to the attackers' site or something else. In any case, the attack script needs to make a connection to an attack site.

If there is a detection algorithm for external resources to the blacklisted site list, it can be suspicious that the website may have a malware. Therefore, any hits against blacklisted sites will indicate the presence of malicious content on a page being analysed.

IV.4 Detection of obfuscating techniques

If a website includes the page content encoded with obfuscating techniques, it could be a reasonable indicator that the website has a malicious purpose. For instance, if a website has content with a long encoded string, it could be a malicious content. However, although the long encoded string is suspicious, it cannot always be assumed that the website has a malicious content until it is decoded and its action is analysed.

IV.5 Evaluation of suspicious content behaviour

The most efficient way is to analyse the behaviour of suspicious content. If the content's activity is suspicious, it can be an indicator of malicious intent. The typical behaviours that could be regarded as malicious include accessing the local hard drive, instantiating a shell application object and downloading (accessing) external executable content.

Appendix V

Typical examples of application security risks by OWASP

(This appendix does not form an integral part of this Recommendation.)

The Open Web Application Security Project (OWASP), an open-source collaboration of web-based security tools, technologies and methodologies from industry leaders, educational organizations and individuals from around the world, published the OWASP top 10 successful web-based attacks [b-OWASP] and common weakness enumeration (CWE) [b-ITU-T X.1524] CWE-928: Weaknesses in OWASP top ten [b-CWE] as shown in Table V.1.

Table V.1 – OWASP top 10 application security risks

Type of attack	Threat agent	Attack vector	Security weakness	Technical impact	Business impact	References to the CWE identifier
A-1 – Injection	Anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attackers send simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources.	Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code, often found in SQL queries, lightweight directory access protocol (LDAP) queries, XPath queries, operating system (OS) commands, program arguments, etc. Injection flaws are easy to discover when examining code, but more difficult via testing. Scanners and fuzzers can help attackers find them.	Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.	Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed?	CWE-77, CWE-78, CWE-89, CWE-90, CWE-91, CWE-929

Table V.1 – OWASP top 10 application security risks

Type of attack	Threat agent	Attack vector	Security weakness	Technical impact	Business impact	References to the CWE identifier
A-2 – Broken authentication and session management	Consider anonymous external attackers, as well as users with their own accounts, who may attempt to steal accounts from others. Also, consider insiders wanting to disguise their actions.	Attacker uses leaks or flaws in the authentication or session management functions (e.g., exposed accounts, passwords, session IDs) to impersonate users.	Developers frequently build custom authentication and session management schemes, but building these correctly is hard. As a result, these custom schemes frequently have flaws in areas such as logout, password management, timeouts, remember me, secret question, account update, etc. Finding such flaws can sometimes be difficult, as each implementation is unique.	Such flaws may allow some or even all accounts to be attacked. Once successful, the attacker can do anything the victim could do. Privileged accounts are frequently targeted.	Consider the business value of the affected data or application functions. Also, consider the business impact of public exposure of the vulnerability.	CWE-256, CWE-287, CWE-384, CWE-311, CWE-319, CWE-522, CWE-523, CWE-613, CWE-620, CWE-640, CWE-930

Table V.1 – OWASP top 10 application security risks

Type of attack	Threat agent	Attack vector	Security weakness	Technical impact	Business impact	References to the CWE identifier
A-3 – Cross-site scripting (XSS)	Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends text-based attack scripts that exploit the interpreter in the browser. Almost any source of data can be an attack vector, including internal sources such as data from the database.	XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. There are three known types of XSS flaws: 1) Stored, 2) Reflected, and 3) DOM based XSS. Detection of most XSS flaws is fairly easy via testing or code analysis.	Attackers can execute scripts in a victim's browser to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc.	Consider the business value of the affected system and all the data it processes. Also consider the business impact of public exposure of the vulnerability.	CWE-79, CWE-931
A-4 – Insecure direct object references	Consider the types of users of your system. Do any users have only partial access to certain types of system data?	Attacker, who is an authorized system user, simply changes a parameter value that directly refers to a system object to another object the user is not authorized for. Is access granted?	Applications frequently use the actual name or key of an object when generating web pages. Applications do not always verify the user is authorized for the target object. This results in an insecure direct object reference flaw. Testers can easily manipulate parameter values to detect such flaws. Code analysis quickly shows whether authorization is properly verified.	Such flaws can compromise all the data that can be referenced by the parameter. Unless object references are unpredictable, it is easy for an attacker to access all available data of that type.	Consider the business value of the exposed data. Also consider the business impact of public exposure of the vulnerability	CWE-22, CWE-99, CWE-639, CWE-932

Table V.1 – OWASP top 10 application security risks

Type of attack	Threat agent	Attack vector	Security weakness	Technical impact	Business impact	References to the CWE identifier
A-5 – Security misconfigura tion	Consider anonymous external attackers as well as users with their own accounts that may attempt to compromise the system. Also, consider insiders wanting to disguise their actions.	Attacker accesses default accounts, unused pages, unpatched flaws, unprotected files and directories, etc. to gain unauthorized access to or knowledge of the system.	Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework and custom code. Developers and system administrators need to work together to ensure that the entire stack is configured properly. Automated scanners are useful for detecting missing patches, misconfigurations, use of default accounts, unnecessary services, etc.	The system could be completely compromised without you knowing it. All of your data could be stolen or modified slowly over time. Recovery costs could be expensive	The system could be completely compromised without you knowing it. All your data could be stolen or modified slowly over time. Recovery costs could be expensive.	CWE-2, CWE-16, CWE-209, CWE-215, CWE-548, CWE-933

Table V.1 – OWASP top 10 application security risks

Type of attack	Threat agent	Attack vector	Security weakness	Technical impact	Business impact	References to the CWE identifier
A-6 – Sensitive data exposure	Consider who can gain access to your sensitive data and any backups of that data. This includes the data at rest, in transit, and even in your customers' browsers. Include both external and internal threats.	Attackers typically do not break crypto directly. They break something else, such as steal keys, do man-in-the-middle (MITM) attacks, or steal clear text data off the server, while in transit, or from the user's browser.	The most common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm usage is common, particularly weak password hashing techniques. Browser weaknesses are very common and easy to detect, but hard to exploit on a large scale. External attackers have difficulty detecting server side flaws due to limited access and they are also usually hard to exploit.	Failure frequently compromises all data that should have been protected. Typically, this information includes sensitive data such as health records, credentials, personal data, credit cards, etc.	Consider the business value of the lost data and impact to your reputation. What is your legal liability if this data is exposed? Also, consider the damage to your reputation.	CWE-310, CWE 311, CWE-312, CWE-319, CWE-325, CWE-326, CWE-934

Table V.1 – OWASP top 10 application security risks

Type of attack	Threat agent	Attack vector	Security weakness	Technical impact	Business impact	References to the CWE identifier
A-7 – Function level access control	Anyone with network access can send your application a request. Could anonymous users access private functionality or regular users a privileged function?	Attacker, who is an authorized system user, simply changes the URL or a parameter to a privileged function. Is access granted? Anonymous users could access private functions that are not protected.	Applications do not always protect application functions properly. Sometimes, function level protection is managed via configuration and the system is misconfigured. Sometimes, developers must include the proper code checks, and they forget. Detecting such flaws is easy. The hardest part is identifying which pages (URLs) or functions exist to attack.	Such flaws allow attackers to access unauthorized functionality. Administrative functions are key targets for this type of attack.	Consider the business value of the exposed functions and the data they process. Also, consider the impact to your reputation if this vulnerability became public.	CWE 285, CWE-287, CWE-935

Table V.1 – OWASP top 10 application security risks

Type of attack	Threat agent	Attack vector	Security weakness	Technical impact	Business impact	References to the CWE identifier
A-8 – Cross-site request forgery (CSRF)	Consider anyone who can load content into your users' browsers, and thus force them to submit a request to your website. Any website or other HTML feed that your users access could do this.	Attacker creates forged HTTP requests and tricks a victim into submitting them via image tags, XSS or numerous other techniques. If the user is authenticated, the attack succeeds.	CSRF takes advantage of the fact that most web applications allow attackers to predict all the details of a particular action. Because browsers send credentials like session cookies automatically, attackers can create malicious web pages which generate forged requests that are indistinguishable from legitimate ones. Detection of CSRF flaws is fairly easy via penetration testing or code analysis.	Attackers can trick victims into performing any state changing operation the victim is authorized to perform, e.g., updating account details, making purchases, logout and even login.	Consider the business value of the affected data or application functions. Imagine not being sure if users intended to take these actions. Consider the impact to your reputation.	CWE-346, CWE-352, CWE-441, CWE-642, CWE-935
A-9 – Using components with known vulnerabilitie s	Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools, expanding the threat agent pool beyond targeted attackers to include chaotic actors.	Attacker identifies a weak component through scanning or manual analysis. He customizes the exploit as needed and executes the attack. It gets more difficult if the used component is deep in the application.	Virtually every application has these issues because most development teams do not focus on ensuring that their components/libraries are up to date. In many cases, the developers do not even know all the components they are using, never mind their versions. Component dependencies make things even worse.	The full range of weaknesses is possible, including injection, broken access control, XSS, etc. The impact could range from minimal to complete host takeover and data compromise.	Consider what each vulnerability might mean for the business controlled by the affected application. It could be trivial or it could mean complete compromise.	CWE-937

Table V.1 – OWASP top 10 application security risks

Type of attack	Threat agent	Attack vector	Security weakness	Technical impact	Business impact	References to the CWE identifier
A-10 – Unvalidated redirects and forwards	Consider anyone who can trick your users into submitting a request to your website. Any website or other HTML feed that your users use could do this.	Attacker links to unvalidated redirect and tricks victims into clicking it. Victims are more likely to click on it, since the link is to a valid site. Attacker targets unsafe forward to bypass security checks.	Applications frequently redirect users to other pages, or use internal forwards in a similar manner. Sometimes, the target page is specified in an unvalidated parameter, allowing attackers to choose the destination page. Detecting unchecked redirects is easy. Look for redirects where you can set the full URL. Unchecked forwards are harder, because they target internal pages.	Such redirects may attempt to install malware or trick victims into disclosing passwords or other sensitive information. Unsafe forwards may allow access control bypass.	Consider the business value of retaining your users' trust. What if they get owned by malware? What if attackers can access internal only functions	CWE-601, CWE-938

Bibliography

[b-ITU-T M.3030] Recommendation ITU-T M.3030 (2002), *Telecommunications Markup Language (tML) framework.*

[b-ITU-T T.411] Recommendation ITU-T T.411 (1993) | ISO/IEC 8613-1:1994, *Information technology − Open Document Architecture (ODA) and interchange format: Introduction and general principles.*

[b-ITU-T X.800] Recommendation ITU-T X.800 (1991), *Security architecture for Open Systems Interconnection for CCITT applications*.

[b-ITU-T X.810] Recommendation ITU-T X.810 (1995) | ISO/IEC 10181-1:1996, *Information technology − Open Systems Interconnection − Security frameworks for open systems: Overview*.

[b-ITU-T X.1252] Recommendation ITU-T X.1252 (2010), *Baseline identity management terms and definitions.*

[b-ITU-T X.1520] Recommendation ITU-T X.1520 (2014), *Common vulnerabilities and exposures.*

[b-ITU-T X.1524] Recommendation ITU-T X.1524 (2012), *Common weakness enumeration.*

[b-ITU-T X.1541] Recommendation ITU-T X.1541 (2012), *Incident object description exchange format.*

[b-ITU-T X.1544] Recommendation ITU-T X.1544 (2013), *Common attack pattern enumeration and classification.*

[b-ITU-T X.1546] Recommendation ITU-T X.1546 (2014), *Malware attribute enumeration and characterization*.

[b-ISO/IEC 27000] ISO/IEC 27000:2014, *Information technology – Security techniques – Information security management systems – Overview and vocabulary*.

[b-ISO/IEC 27033-1] ISO/IEC 27033-1:2009, *Information technology – Security techniques – Network security – Part 1: Overview and concepts.*

[b-CAPEC-62] CAPEC-62: *Cross Site Request Forgery (aka Session Riding).*
https://capec.mitre.org/data/definitions/62.html

[b-CAPEC-66] CAPEC-66: *SQL Injection*.
https://capec.mitre.org/data/definitions/66.html

[b-CAPEC-103] CAPEC-103: *Clickjacking.*
https://capec.mitre.org/data/definitions/103.html

[b-CWE] CWE-928: *Weaknesses in OWASP Top Ten (2013).*
http://cwe.mitre.org/data/graphs/928.html

[b-iframe] W3C (2014), *HTML <iframe> Tag*.
http://www.w3schools.com/tags/tag_iframe.asp

[b-NA] Idika, Nwokedi, and Mathur, Aditya P. (2007), *A Survey of Malware Detection Techniques,* Department of Computer Science, Purdue University, 2 February.
http://www.serc.net/system/files/SERC-TR-286.pdf

[b-NIST SP 800-83]	NIST Special Publication 800-83 (2005), *Guide to Malware Incident Prevention and Handling.*
[b-NTobjectives]	Kuykendall, Dan (2009), *Is Your Website Already Infected? Analyzing and Detecting Malicious Content,* 20 March. http://www.manvswebapp.com/is-your-website-already-infected
[b-OWASP]	OWASP (2013), *OWASP Top 10 application security risks*. https://www.owasp.org/index.php/Top_10_2013-Top_10

Printed in Switzerland
Geneva, 2014