Figure 1: OWASP LOGO

OWASP Top 10 2017 RC1++

[date]

Release Candidate - Important Notice

OWASP plans to release the final public release of the OWASP Top 10 - 2017 in July or August 2017 after a public comment period ending June 30, 2017.

This release of the OWASP Top 10 marks this project's fourteenth year of raising awareness of the importance of application security risks. This release follows the 2013 update, whose main change was the addition of 2013-A9 Use of Known Vulnerable Components. We are pleased to see that since the 2013 Top 10 release, a whole ecosystem of both free and commercial tools have emerged to help combat this problem as the use of open source components has continued to rapidly expand across practically every programming language. The data also suggests the use of known vulnerable components is still prevalent, but not as widespread as before. We believe the awareness of this issue the Top 10 - 2013 generated has contributed to both of these changes.

We also noticed that since CSRF was introduced to the Top 10 in 2007, it has dropped from a widespread vulnerability to an uncommon one. Many frameworks include automatic CSRF defenses which has significantly contributed to its decline in prevalence, along with much higher awareness with developers that they must protect against such attacks.

Constructive comments on this OWASP Top 10 - 2017 Release Candidate should be forwarded via email to OWASP-TopTen@lists.owasp.org. Private comments may be sent to dave.wichers@owasp.org. Anonymous comments are welcome. All non-private comments will be catalogued and published at the same time as the final public release. Comments recommending changes to the items listed in the Top 10 should include a complete suggested list of 10 items, along with a rationale for any changes. All comments should indicate the specific relevant page and section.

Following the final publication of the OWASP Top 10 - 2017, the collaborative work of the OWASP community will continue with updates to supporting documents including the OWASP wiki, OWASP Developer's Guide, OWASP Testing Guide, OWASP Code Review Guide, and the OWASP Prevention Cheat Sheets, along with translations of the Top 10 to many different languages.

Your feedback is critical to the continued success of the OWASP Top 10 and all other OWASP Projects. Thank you all for your dedication to improving the security of the world's software for everyone.

- Andrew van der Stock, vanderaj@owasp.org (Project Lead 2007, 2017)
- Dave Wichers, dave.wichers@owasp.org (Project Lead 2003-2017)
- Jeff Williams, jeff.williams@owasp.org (Project Lead 2003-2017)

Introduction

Welcome

Welcome to the OWASP Top 10 2017! This major update adds two new vulnerability categories for the first time: (1) Insufficient Attack Detection and Prevention and (2) Underprotected APIs. We made room for these two new categories by merging the two access control categories (2013-A4 and 2013-A7) back into Broken Access Control (which is what they were called in the OWASP Top 10 - 2004), and dropping 2013-A10: Unvalidated Redirects and Forwards, which was added to the Top 10 in 2010.

The OWASP Top 10 for 2017 is based primarily on 11 large datasets from firms that specialize in application security, including 8 consulting companies and 3 product vendors. This data spans vulnerabilities gathered from hundreds of organizations and over 50,000 real-world applications and APIs. The Top 10 items are selected and prioritized according to this prevalence data, in combination with consensus estimates of exploitability, detectability, and impact.

The primary aim of the OWASP Top 10 is to educate developers, designers, architects, managers, and organizations about the consequences of the most important web application security weaknesses. The Top 10 provides basic techniques to protect against these high risk problem areas – and also provides guidance on where to go from here.

Warnings

Don't stop at 10. There are hundreds of issues that could affect the overall security of a web application as discussed in the OWASP Developer's Guide and the OWASP Cheat Sheet Series. These are essential reading for anyone developing web applications and APIs. Guidance on how to effectively find vulnerabilities in web applications and APIs is provided in the OWASP Testing Guide and the OWASP Code Review Guide.

Constant change. This Top 10 will continue to change. Even without changing a single line of your application's code, you may become vulnerable as new flaws are discovered and attack methods are refined. Please review the advice at the end of the Top 10 in "What's Next For Developers, Verifiers, and Organizations" for more information.

Think positive. When you're ready to stop chasing vulnerabilities and focus on establishing strong application security controls, OWASP is maintaining and promoting the Application Security Verification Standard (ASVS) as a guide to organizations and application reviewers on what to verify.

Use tools wisely. Security vulnerabilities can be quite complex and buried in mountains of code. In many cases, the most cost-effective approach for finding and eliminating these weaknesses is human experts armed with good tools.

Push left, right, and everywhere. Focus on making security an integral part of your culture throughout your development organization. Find out more in the OWASP Software Assurance Maturity Model (SAMM) and the Rugged Handbook.

Attribution

Thanks to Aspect Security for initiating, leading, and updating the OWASP Top 10 since its inception in 2003, and to its primary authors: Jeff Williams and Dave Wichers.

We'd like to thank the many organizations that contributed their vulnerability prevalence data to support the 2017 update, including these large data set providers:

Aspect Security, AsTech Consulting, Branding Brand, Contrast Security, EdgeScan, iBLISS, Minded Security, Paladion Networks, Softtek, Vantage Point, Veracode

For the first time, all the data contributed to a Top 10 release, and the full list of contributors, is publicly available.

We would like to thank in advance those who contribute significant constructive comments and time reviewing this update to the Top 10 and to:

- Neil Smithline – Generating the Wiki version
- Torsten Gigler - German translation

And finally, we'd like to thank in advance all the translators out there that will translate this release of the Top 10 into numerous different languages, helping to make the OWASP Top 10 more accessible to the entire planet.

Copyright and License

Figure 2: license

Project Leads	Lead Authors	Contributors and Reviewers
Andrew van der Stock	Dave Wichers, Jeff Williams	TBA

About OWASP

The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications and APIs that can be trusted. At OWASP you'll find free and open …

Application security tools and standards Complete books on application security testing, secure code development, and secure code review Standard security controls and libraries Local chapters worldwide Cutting edge research Extensive conferences worldwide Mailing lists

Learn more at: https://www.owasp.org

All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security. We advocate approaching application security as a people, process, and technology problem, because the most effective approaches to application security require improvements in all of these areas.

OWASP is a new kind of organization. Our freedom from commercial pressures allows us to provide unbiased, practical, cost-effective information about application security. OWASP is not affiliated with any technology company, although we support the informed use of commercial security technology. Similar to many open source software projects, OWASP produces many types of materials in a collaborative, open way.

The OWASP Foundation is the non-profit entity that ensures the project's long-term success. Almost everyone associated with OWASP is a volunteer, including the OWASP Board, Chapter Leaders, Project Leaders, and project members. We support innovative security research with grants and infrastructure.

Come join us!

A1 - Injections

1	2	3	4	5
Application Specific	Exploitability Easy	Prevalence Common	Impact Severe	Application Business Specific

1	2	3	4	5
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, 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. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers,		

1	2	3	4	5

Am I vulnerable to	How do I prevent
The best way to find out if an application is vulnerable to injection is to verify that all use of interpreters clearly separates untrusted data from the command or query. In many cases,	

Am I vulnerable to	How do I prevent

Checking the code is a fast and accurate way to see if the application uses interpreters safely. Code analysis tools can help a security analyst find use of interpreters and trace data flow through the application. Penetration testers can validate these issues by crafting exploits that confirm the vulnerability. Automated dynamic scanning which exercises the application may provide insight into whether some exploitable injection flaws exist. Scanners cannot always reach interpreters and have difficulty detecting whether an attack was successful. Poor error handling makes injection flaws easier to discover. | Preventing injection requires keeping untrusted data separate from commands and queries. The preferred option is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface. Be careful with APIs, such as stored procedures, that are parameterized, but can still introduce injection under the hood. If a parameterized API is not available, you should carefully escape special characters using the specific escape syntax for that interpreter. OWASP's Java Encoder and similar libraries provide such escaping routines. Positive or "white list" input validation is also recommended, but is not a complete defense as many situations require special characters be allowed. If special characters are required, only approaches (1) and (2) above will make their use safe. OWASP's ESAPI has an extensible library of white list input validation routines |

Example Scenarios	References
tba	tba

A1 - Injections

1	2	3	4	5
Application Specific	Exploitability Easy	Prevalence Common	Impact Severe	Application Business Specific

1	2	3	4	5
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, 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. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers,		

1	2	3	4	5

Am I vulnerable to	How do I prevent
tba	tba

Example Scenarios	References
tba	tba

A1 - Injections

1	2	3	4	5
Application Specific	Exploitability Easy	Prevalence Common	Impact Severe	Application Business Specific

1	2	3	4	5
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, 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. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers,		

1	2	3	4	5

Am I vulnerable to	How do I prevent
tba	tba

Example Scenarios	References
tba	tba

A1 - Injections

1	2	3	4	5
Application Specific	Exploitability Easy	Prevalence Common	Impact Severe	Application Business Specific

1	2	3	4	5
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, 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. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers,		

1	2	3	4	5

Am I vulnerable to	How do I prevent
tba	tba

Example Scenarios	References
tba	tba

A1 - Injections

1	2	3	4	5
Application Specific	Exploitability Easy	Prevalence Common	Impact Severe	Application Business Specific

1	2	3	4	5
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, 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. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers,		

1	2	3	4	5

Am I vulnerable to	How do I prevent
tba	tba

Example Scenarios	References
tba	tba

A1 - Injections

1	2	3	4	5
Application Specific	Exploitability Easy	Prevalence Common	Impact Severe	Application Business Specific

1	2	3	4	5
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, 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. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers,		

1	2	3	4	5

Am I vulnerable to	How do I prevent
tba	tba

Example Scenarios	References
tba	tba

A1 - Injections

1	2	3	4	5
Application Specific	Exploitability Easy	Prevalence Common	Impact Severe	Application Business Specific

1	2	3	4	5
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, 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. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers,		

1	2	3	4	5

Am I vulnerable to	How do I prevent
tba	tba

Example Scenarios	References
tba	tba

A1 - Injections

1	2	3	4	5
Application Specific	Exploitability Easy	Prevalence Common	Impact Severe	Application Business Specific

1	2	3	4	5
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, 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. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers,		

1	2	3	4	5

Am I vulnerable to	How do I prevent
tba	tba

Example Scenarios	References
tba	tba

A1 - Injections

1	2	3	4	5
Application Specific	Exploitability Easy	Prevalence Common	Impact Severe	Application Business Specific

1	2	3	4	5
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, 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. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers,		

1	2	3	4	5

Am I vulnerable to	How do I prevent
tba	tba

Example Scenarios	References
tba	tba

A1 - Injections

1	2	3	4	5
Application Specific	Exploitability Easy	Prevalence Common	Impact Severe	Application Business Specific

1	2	3	4	5
Consider anyone who can send untrusted data to the system, including external users, business partners, other systems, 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. They are often found in SQL, LDAP, XPath, or NoSQL queries; OS commands; XML parsers, SMTP Headers,		

1	2	3	4	5

Am I vulnerable to	How do I prevent
tba	tba

Example Scenarios	References
tba	tba

Appendix A: Glossary

- **2FA** – Two-factor authentication(2FA) adds a second level of authentication to an account log-in.
- **Address Space Layout Randomization (ASLR)** – A technique to make exploiting memory corruption bugs more difficult.
- **Application Security** – Application-level security focuses on the analysis of components that comprise the application layer of the Open Systems Interconnection Reference Model (OSI Model), rather than focusing on for example the underlying operating system or connected networks.
- **Application Security Verification** – The technical assessment of an application against the OWASP MASVS.
- **Application Security Verification Report** – A report that documents the overall results and supporting analysis produced by the verifier for a particular application.
- **Authentication** – The verification of the claimed identity of an application user.
- **Automated Verification** – The use of automated tools (either dynamic analysis tools, static analysis tools, or both) that use vulnerability signatures to find problems.
- **Black box testing** – It is a method of software testing that examines the functionality of an application without peering into its internal structures or workings.
- **Component** – a self-contained unit of code, with associated disk and network interfaces that communicates with other components.
- **Cross-Site Scripting** (XSS) – A security vulnerability typically found in web applications allowing the injection of client-side scripts into content.
- **Cryptographic module** – Hardware, software, and/or firmware that implements cryptographic algorithms and/or generates cryptographic keys.
- **DAST** –Dynamic application security testing (DAST) technologies are designed to detect conditions indicative of a security vulnerability in an application in its running state.

- **Design Verification** – The technical assessment of the security architecture of an application.
- **Dynamic Verification** – The use of automated tools that use vulnerability signatures to find problems during the execution of an application.
- **Globally Unique Identifier** (GUID) – a unique reference number used as an identifier in software.
- **Hyper Text Transfer Protocol** (HTTP) – An application protocol for distributed, collaborative, hypermedia information systems. It is the foundation of data communication for the World Wide Web.
- **Hardcoded keys** – Cryptographic keys which are stored in the device itself.
- **IPC** – Inter Process Communications,In IPC Processes communicate with each other and with the kernel to coordinate their activities.
- **Input Validation** – The canonicalization and validation of untrusted user input.
- **JAVA Bytecode -** Java bytecode is the instruction set of the Java virtual machine(JVM). Each bytecode is composed of one, or in some cases two bytes that represent the instruction (opcode), along with zero or more bytes for passing parameters.
- **Malicious Code** – Code introduced into an application during its development unbeknownst to the application owner, which circumvents the application's intended security policy. Not the same as malware such as a virus or worm!
- **Malware** – Executable code that is introduced into an application during runtime without the knowledge of the application user or administrator.
- **Open Web Application Security Project** (OWASP) – The Open Web Application Security Project (OWASP) is a worldwide free and open community focused on improving the security of application software. Our mission is to make application security "visible," so that people and organizations can make informed decisions about application security risks. See: http://www.owasp.org/
- **Personally Identifiable Information** (PII) - is information that can be used on its own or with other information to identify, contact, or locate a single person, or to identify an individual in context.
- **PIE** – Position-independent executable (PIE) is a body of machine code that, being placed somewhere in the primary memory, executes properly regardless of its absolute address.
- **PKI** – A PKI is an arrangement that binds public keys with respective identities of entities. The binding is established through a process of registration and issuance of certificates at and by a certificate authority (CA).
- **SAST** – Static application security testing (SAST) is a set of technologies designed to analyze application source code, byte code and binaries for coding and design conditions that are indicative of security vulnerabilities. SAST solutions analyze an application from the "inside out" in a nonrunning state.

- **SDLC** – Software development lifecycle.
- **Security Architecture** – An abstraction of an application's design that identifies and describes where and how security controls are used, and also identifies and describes the location and sensitivity of both user and application data.
- **Security Configuration** – The runtime configuration of an application that affects how security controls are used.
- **Security Control** – A function or component that performs a security check (e.g. an access control check) or when called results in a security effect (e.g. generating an audit record).
- **SQL Injection (SQLi)** – A code injection technique used to attack data driven applications, in which malicious SQL statements are inserted into an entry point.
- **SSO Authentication** – Single Sign On(SSO) occurs when a user logs in to one Client and is then signed in to other Clients automatically, regardless of the platform, technology, or domain the user is using. For example when you log in in google you automatically login in the youtube , docs and mail service.
- **Threat Modeling -** A technique consisting of developing increasingly refined security architectures to identify threat agents, security zones, security controls, and important technical and business assets.
- **Transport Layer Security** – Cryptographic protocols that provide communication security over the Internet
- **URI/URL/URL fragments** – A Uniform Resource Identifier is a string of characters used to identify a name or a web resource. A Uniform Resource Locator is often used as a reference to a resource.
- **User acceptance testing (UAT)**– Traditionally a test environment that behaves like the production environment where all software testing is performed before going live.
- **Verifier** – The person or team that is reviewing an application against the OWASP ASVS requirements.
- **Whitelist** – A list of permitted data or operations, for example a list of characters that are allowed to perform input validation.
- **X.509 Certificate** – An X.509 certificate is a digital certificate that uses the widely accepted international X.509 public key infrastructure (PKI) standard to verify that a public key belongs to the user, computer or service identity contained within the certificate.

Appendix B: References

The following OWASP projects are most likely to be useful to users/adopters of this standard:

- OWASP Proactive Controls -https://www.owasp.org/index.php/

OWASP_Proactive_Controls

- OWASP Application Security Verification Standard -https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project
- OWASP Privacy Top 10 Risks -https://www.owasp.org/index.php/OWASP_Top_10_Privacy_Risks_Project
- OWASP Mobile Top 10 Risks -https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks

Similarly, the following web sites are most likely to be useful to users/adopters of this standard:

- MITRE Common Weakness Enumeration - http://cwe.mitre.org/
- PCI Security Standards Council - https://www.pcisecuritystandards.org
- PCI Data Security Standard (DSS) v3.0 Requirements and Security Assessment Procedures https://www.pcisecuritystandards.org/documents/PCI_DSS_v3.pdf