A Multidimensional View of Critical Web Application Security Risks: A Novel ‘Attacker-Defender’ PoV

By assessing specific functional attributes across the application and IT architecture, security and remediation teams can more effectively anticipate and plug vulnerabilities exploited by hackers and other cybercriminals.
EXECUTIVE SUMMARY

Web applications have evolved to meet a wide range of business requirements. The increasing complexity of these applications significantly augments the attack surface of the infrastructure and thus leaves an organization open to potential security threats. With the various user-interactive functionalities such as login, registration, payment, etc. that deal with underlying components such as databases, lightweight directory access protocol (LDAP) repositories greatly increase the attack surface area and become prime areas of focus for hackers. These functionalities act as entry and exit points to the application and underlying infrastructure.

Successful penetration through the application layer leads to attacks that may cause remote code execution with web server privileges, unauthorized access to information stored in a web server, website content defacement, deletion of files in the web server and denial of service. Any of these outcomes can damage the organization’s reputation. The fundamental security problem with web applications is that all user input is considered untrusted; this requires the implementation of a number of security mechanisms to enable applications to defend themselves against attack. Figure 1 (see next page) depicts vulnerability distribution for 2016 across different verticals.
This white paper illuminates various defense mechanisms that can be applied to key functionalities of a web application to secure them from being attacked. Based on our project engagements in the year 2016, the vulnerability distribution across the different functionalities are highlighted in Figure 2 (see next page).

Our attacker-defender approach considers common functionalities in any web application from an attacker’s perspective and then presents the defense techniques to be employed in order to secure the application. The following sections elucidate the attacker and defender approach for different functionalities commonly found in a web application.

**LOGIN/LOGOUT**

User authenticity has become a necessity in almost every web application and is typically managed through the login and logout functionalities. These functionalities are the front line of defense for an application and are seen in every small to complex web application owned by different industry sectors such as banking, healthcare and retail. Authentication services limit unauthorized users in conjunction with certain other protected features of the application. Authentication functionalities such as login and logout, in our view, are more often subject to design weaknesses than any other security mechanisms employed in web applications. Authentication technologies vary from HTML form-based authentication, multifactorial mechanisms such as combining passwords and physical tokens, client secure socket layer (SSL) certifications and smartcards, HTTP basic and digest authentication, and Windows integrated authentication using NTLM or Kerberos protocols.

**Attacker-Defender Approach**

To gain a cohesive understanding of application security issues, a slightly modified attack tree can be deployed. The tree will represent several possible attacks that are targeted at a specific functionality, along with the corresponding...
mitigation techniques to hamper the attack. Figure 3 (see next page) depicts such an attack tree for login/logout functionality, which includes attack methods and attacks that aim to gain user credentials. The tree also enlists the remediation methods to defend against the attacks.

Injection

- **Attacker**: Injection flaws - such as SQL, SQLi, bSQLi, NoSQLi, HQL injection and LDAP injection - occur when untrusted data is sent to an interpreter as part of a command or query. The attacker usually sends simple text-based messages that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources.

- **Defender**: The most effective way to prevent injection attacks is to use parameterized queries (prepared statements) for all database access. This two-step method incorporates potentially tainted data into all types of SQL queries: first, the application specifies the structure of the query, leaving placeholders for each item of user input; second, the application specifies the contents of each placeholder. Because the structure of the query has already been defined in the first step, it is not possible for malformed data in the second step to interfere with the query structure. One of the most powerful controls, if done well, is validation of the input that an application receives. It can be as simple as strictly typing a parameter and as complex as using regular expressions or business logic to validate input.

There are two different types of input validation approaches: whitelist validation (inclusion or positive validation) and blacklist validation (exclusion or negative validation).

Phishing Through Frames

- **Attacker**: Phishing is a scenario which involves an e-mail message that asks users to update their personal information with a link to a spoofed website. Frames are a popular method of hiding attack content due to their uniform browser support and easy coding style. The page linked to within the hidden frame can be used to deliver additional content, retrieve confidential information such as session IDs, or do something more elaborate such as executing screen-grabbing and key-logging while the user is exchanging
**Attack Tree with Mitigations for Login/Logout Functionality Vulnerabilities**

- **FUNCTIONALITY**
  - Login/Logout Functionality
  - Gaining User Credentials Information

- **ATTACK METHODS**
  - Attacks Related to User Interface
    - Password replay
    - Browser refresh attack
    - Click-jacking
    - SQL injection
  - Attacks Related to N/W Channel
    - Getting the username/password value from history (get method enabled)
    - Enabling the browser configuration to get sensitive user data (Autocomplete set=ON)
    - Retrieving sensitive data from temp file (Http page enabled the cache/store)
    - Sniffing the unencrypted data channel
    - Weak SSL ciphers would permit decrypting and intercepting a particular SSL
    - Forgery of the self-signed certificate

- **ATTACKS**
  - Application should redirect to generic error page.
  - Hash the password before the data is sent to the server.
  - Embed click-jacking defensive code in the UI window.
  - Username/password should be passed on the POST request.
  - Enable the no-cache/no-store flag.
  - Proper SSL certification should be enabled with standard cryptographic algorithms.
  - Standard cryptographic algorithms (NIST/local policy) should be used to encrypt the sensitive data.

- **MITIGATION**

**Figure 3**
confidential information over the Internet. Through this attack, an adversary can trick the user into entering the login credentials to a spoofed website and capture the content in a hidden frame.

**Defender:** There are two modes of defending mechanism for this attack:

- **Browser perspective:** Browser pop-ups are a common attack technique used by attackers to make it appear that the requests are coming from a victim domain. Disabling pop-ups will make it much more difficult for attackers to take over the user’s session without being detected.

- **Application coding perspective:** As attackers use frames to host malicious content, they can discover the confidential information in the application. The best practice here is to use a Target directive to create a new window that will usually break out of an iframe and other JavaScript jails.

**Attacker-Defender Approach**

The two major attack methods for stealing payment information are UI-related and network-related attacks. Attacks that are executed through the user interface include injection, accessing clipboard data, cross-site scripting, cross-site request forgery (CSRF), etc. Network-related attacks, however, are accomplished over the network channel - e.g., sniffing, decrypted weak ciphers and self-signed SSL certificate forgery.

**Cross-Site Scripting**

- **Attacker:** Cross-site scripting (XSS) is a type of injection problem in which malicious scripts are injected into a trusted website. XSS flaws occur whenever an application sends untrusted data without validation or encoding to a web browser or stores it in the target servers.

- **Defender:** XSS can be prevented by performing proper input validation and output encoding on both the client and server sides such that the scripts are not executable. Filter out the hazardous characters from the user input into the web application.

**CSRF**

- **Attacker:** The attacker can force the user to send unintended requests to the application server and perform malicious actions on behalf of the web application user who has already logged into the application.

- **Defender:** Insert custom random tokens into every form and URL that will not be automatically submitted by the browser. Every request should contain a unique identifier,
Do not use GET requests (URLs); instead, use POST when processing sensitive data requests.

**Attack Tree with Mitigations for Payment-Related Functionality Vulnerabilities**

**Payment Functionality**

**Gaining Sensitive User Payment Information**

**ATTACK METHODS**

- Attacks Related to User Interface
  - Cross-site scripting
  - SQL injection
  - CSRF

- Attacks Related to N/W Channel
  - Bypass the nonstandard cryptographic algorithm using known plain text, cipher text attack
  - Session-based attacks (session hijacking/session fixation)
  - Retrieving sensitive data from temp file (Https page enabled the cache/store)
  - Sniffing the unencrypted data channel
  - Decrypt the SSL certificate if weak cipher enabled in the application
  - Forgery of the self-signed certificate

**MITIGATION**

- White list validation should apply to all the user-controlled data.
- Output encoding should apply to the server response.
- Escape the malicious characters.
- White list validation should apply to all the user-controlled data.
- Parameterized data passed to the application. Application should not display the detailed error message.
- Pass the unique token value to each request.
- Enable the no-cache/no-store flag.
- Standard cryptographic algorithms (NIST/local policy) should be used to encrypt the sensitive data.
- Session value is properly invalidated at server side.
- Unique token value should be used in each session.
- Implement secure session management. Use strong session IDs, protect them in transit and regenerate session identifiers at frequent intervals.
- Proper SSL certification should be enabled with standard cryptographic algorithms.

Figure 4
which is a parameter that an attacker cannot guess. Do not use GET requests (URLs); instead, use POST when processing sensitive data requests.

Retrieving Sensitive Data from a Temp File

- **Attacker:** It is possible for an attacker to gather sensitive information about the payment application such as usernames, passwords, credit card data, account numbers, machine names and/or sensitive file locations.

- **Defender:** Clear all parameters, sensitive information and input values when the page is being loaded/reloaded.

//Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: 0/

Figure 4 (see previous page) depicts an attack tree for payment functionality, comprising attack methods, types of attacks that aim to gain sensitive user payment information and various mitigation techniques.

**SEARCH**

Search functionality is commonly used in most applications to enable users to discover content

---

**Attack Tree with Mitigations for Search-Functionality-Related Vulnerabilities**

**FUNCTIONALITY**

- Search Functionality

**ATTACK METHODS**

- Execution of Unintended Payloads

**ATTACKS**

- XSS
- SQL Injection
- HTTP Response Splitting

**MITIGATION**

- White list validation should apply to all the user-controlled data.
- Output encoding should apply to the server response.
- Escape the malicious characters.

- White list validation should apply to all the user-controlled data.
- Parameterized data passed to the application.
- Application should not display the detailed error message.

- Sanitize the response header when user input is reflected in the response header.

---

Figure 5
contained in a data repository. Search pages are usually constructed with a single form field and a submit button. A search query would display both the matched results and the searched-for text. Attackers often attempt to exploit search functionality behaviors to execute unintended queries or malicious scripts.

**Attacker-Defender Approach**

**HTTP Response Splitting**

- **Attacker:** A response splitting attack is possible only if there is a proxy server used by multiple users to connect to various websites. The attacker will be able to modify the request header with a value and two responses, separated by `%0d%0a` (CRLF) code. Immediately after sending the first request, the attacker sends a second request for a valid publicly accessible page on the site/server.

- **Defender:** Use server side validation and disallow CRLF characters in all requests where user input is reflected in the response header.

Figure 5 (see previous page) depicts an attack tree for search functionality attacks that aim to execute unintended payloads, plus defensive remediation techniques.

**REGISTRATION**

Registration is a basic and essential function. Self-service registration functionality allows new users to register or enroll in the application by providing personal details such as username, date of birth, e-mail address, security questions, etc. The new user is registered if all provided details fit according to the application's requirements, thus allowing users to log in thereafter. Since all users who try to log into the application are not always legitimate users, the application should validate unauthorized inputs before they are processed.

**Attacker-Defender Approach**

**Enumerating User Information**

- **Attacker:** Enumeration is the first stage of the attack; it is the process used to gather the information about a target application by actively connecting to it and identifying the user account, system account and admin account. It is also an activity in which an attacker tries to retrieve valid usernames from a web application. If the system is vulnerable to this attack, the attacker may be able to obtain a list of existing usernames in the system by submitting input (valid and invalid usernames) and analyzing the server response (error messages). The scope of this test is to verify if it is possible to collect a set of valid usernames by interacting with the application's authentication mechanism. The attacker can then run a dictionary attack to further exploit the obtained information.

- **Defender:** The effective way to prevent enumeration attacks is to add CAPTCHA in the registration page. Also, display only the customized error messages to the user interface, and disable the unnecessary comments in the source code to prevent the attacker from gathering information from the error messages.

**Automated Multiple Registration**

- **Attacker:** The attacker tries to increase the size of the request by appending an enormous amount of data that is sent to the server. This could result in a delayed response or server hanging. The attacker can also send
"n" number of requests to the server for registering multiple times to cause the denial of service attacks.

- **Defender:** The most effective way to prevent automated multiple registration is to validate the content length and check for the file size that is being passed in the request. If the content size is more than the specified limit, drop that particular request. If there are too many requests in the queue, then the upcoming request should be automatically dropped without serving. Approaches such as a one-time password, generating QR code and using CAPTCHA riddles should be implemented to reduce the impact of this attack.

### Sniffing the Unencrypted Data Channel

- **Attacker:** This is a type of cyberattack where a malicious user inserts him/herself into a conversation between two parties, impersonates both parties and gains access to information that the two parties were trying to send to each other. This attack allows a malicious user to intercept, send and receive data meant for someone else and gain access to the unauthorized resources.

- **Defender:** Use strong encryption standards between the client and the server; also, the server should authenticate the client’s request by presenting a digital certificate, and only then allow connection to be established.

### Decrypt the SSL Certificate if Weak Cipher Is Enabled in the Application

- **Attacker:** All systems and applications utilizing the SSL with cipher-block chaining mode ciphers may be vulnerable. By decrypting this SSL certificate, an attacker can gain access to sensitive data passed within the encrypted web session, such as passwords, cookies and other authentication tokens. These can then be used to gain more complete access to a website (impersonating that user, accessing database content, etc.).

- **Defender:** It is important to check the SSL configuration being used to avoid putting in place cryptographic support that could be easily defeated. Accordingly, an SSL-based service should not offer the possibility to choose a weak cipher suite. A cipher suite is specified by an encryption protocol (e.g., DES, RC4, AES), the encryption key length (e.g., 256 bits) and a hash algorithm (e.g., SHA, MD5) used for integrity checking.

### Forge the Self-Signed Certificate

- **Attacker:** The attackers usually use self-signed digital certificates or stolen certificates that are accepted as valid by most browsers. The browsers display a warning message when encountering errors during SSL certificate validation, but users can proceed anyway. This is the typical scenario for fake SSL connections, which triggers
Self-signed certificates with pinning are more secure than CA-signed certificates.

**Attack Tree with Mitigations for Registration-Functionality-Related Vulnerabilities**

**FUNCTIONALITY**

- Registration Functionality
  - Gaining Sensitive Information
    - Attacks Related to User Interface
      - Enumerating user information
      - Enabling the browser configuration to get sensitive user data (Autocomplete set=ON)
      - Cross-site scripting
      - SQL injection
    - Attacks Related to N/W Channel
      - Retrieving sensitive data from temp file (Https page enabled the cache/store)
      - Automated multiple registration
      - Sniffing the unencrypted data channel
      - Decrypt the SSL certificate if weak cipher is enabled in the application
      - Forgery of the self-signed certificate

**ATTACK METHODS**

**ATTACKS**

- Enumerating user information
- Enabling the browser configuration to get sensitive user data (Autocomplete set=ON)
- Cross-site scripting
- SQL injection
- Retrieving sensitive data from temp file (Https page enabled the cache/store)
- Automated multiple registration
- Sniffing the unencrypted data channel
- Decrypt the SSL certificate if weak cipher is enabled in the application
- Forgery of the self-signed certificate

**MITIGATION**

- Customized error message should be revealed to the user.
- Unnecessary source code comments should be disabled.
- Autocomplete set=OFF for sensitive fields.
- Whitelist validation should apply to all the user-controlled data.
- Parameterized data passed to the application.
- Application should not display the detailed error message.
- Whitelist validation should apply to all the user-controlled data. Output encoding should apply to the server response.
- Escape the malicious characters.
- Enable the no-cache/no-store flag.
- Proper SSL certification should be enabled with standard cryptographic algorithms.

Figure 6
a certificate warning, caused primarily by server misconfigurations. However, these alerts are often ignored by users who trust forged SSL certificates.

- **Defender:** Browser vendors could mitigate this cyber threat by adopting HTTP Strict Transport Security, Public Key Pinning and TLS Origin Bound Certificates, and by validating certificates with notaries. In general, self-signed certificates with pinning are more secure than CA-signed certificates.

Figure 6 (see previous page) depicts an attack tree for registration functionality, illustrating attack methods and types of attacks that attempt to gain sensitive information from the user. The tree also elaborates several countermeasures.

**FILE UPLOAD/DOWNLOAD**

Uploaded files represent a significant risk to applications. The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems and simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.

**Attacker-Defender Approach**

**Remote File Inclusion (RFI)/Local File Inclusion (LFI)**

- **Attacker:** Remote file inclusion (RFI) is a type of vulnerability most often found on websites. It allows an attacker to include a remote file, usually through a script on the web server. The vulnerability occurs due to the use of user-supplied input without proper validation. An attacker may use streams to exploit RFI vulnerable parameters. RFI attacks are highly automated, judging by traffic shape (e.g., consistency and rate) and characteristics (e.g., distinctive HTTP headers), making them very suitable for mitigation via reputation-based blacklists. By exploiting RFI vulnerability, an attacker can inject a c99 shell to attack a web server. Scripts also can be injected through RFI in order to deface the websites.

In local file inclusion (LFI), which is similar to remote file inclusion vulnerability, only local files (i.e., files on the current server) are included. The vulnerability is also due to the use of user-supplied input without proper validation. LFI enables an attacker to include code that is already hosted on the same web server as the application. LFI vulnerability exploitation requires that the malicious code is hosted on the vulnerable server. By using the presence of LFI, an attacker can execute the remote code via an Apache server log. Code can also be executed via uploading files by including some script files in the uploaded files.

- **Defender:** If the uploaded file needs to be stored on the disk, use a server-generated filename. Inspect the content of uploaded files, and enforce a whitelist of accepted, non-executable content types. Enforce a whitelist of accepted, non-executable file extensions. And also ensure that the file extension matches the actual type of the file content. Use a predefined switch/case statement to determine which file to include rather than using a URL or form parameter to dynamically generate the path. If uploaded files are downloaded by users, provide an accurate non-generic content-type header.
Enforce appropriate authorization on all critical functionalities.

Attack Tree with Mitigations for File-Upload-Related Functionality Vulnerabilities

**File Upload**

**Execution of Unintended File/Shell/Payloads**

- **Remote File Inclusion**
- **Local File Inclusion**
- **Malicious Content Upload (Shell/Batch)**

**MITIGATION**

- Use a server-generated filename if storing uploaded files on disk.
- Inspect the content of uploaded files, and enforce a whitelist of accepted, non-executable content types.
- Enforce a whitelist of accepted, non-executable file extensions.
- If uploaded files are downloaded by users, supply an accurate non-generic content-type header.
- Enforce a size limit on uploaded files. Reject attempts to upload archive formats such as ZIP/war/jar.

**PARAMETER TAMPERING**

**Attacker:** The attacker attempts to change the role of his/her user ID to a higher privileged one. First the attacker identifies the parameter representing the user role that is sent in HTTP requests to the application. The attacker then modifies the parameter to a higher privileged one and gains additional privileges. The attack is possible when the application relies on user role/level parameters in HTTP requests to determine the user’s access level. These parameters could initially be set by the application upon authentication, in the HTTP response as cookies or in hidden fields. Based on the parameter, the application could return a list of application functionalities/menu-items applicable to the user. Since a client-supplied role parameter of the application is accepted

**Privileged User Functionalities**

Privilege escalation attacks aim to obtain additional privileges for web application users to access critical system resources, functions, pages or accounts. They can be either vertical or horizontal privilege escalation attacks.
by the server, it is possible to tamper with these values.

The attacker force-browses into a particular restricted functionality by tampering with HTTP query parameters. For example: If a user is authorized to only view the list of users using www.app.com/users.aspx?fn=view, he may edit or delete users by force-browsing to www.app.com/users.aspx?fn=edit. The attacker retrieves the data of another user by modifying exposed system object references. Then the attacker attempts to obtain other users’ details by modifying the primary key value such as a database table or record value, exposed by the application.

- **Defender:** Do not rely on client-supplied values of user level or role ID to determine the access level for a user. Implement proper access control at the server side for all users. Enforce appropriate authorization on all critical functionalities. Perform authorization checks at the server side to ensure the user is authorized for the requested resource/function. Do not expose references to system objects or primary keys. Each use of an object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object. Use per-user or session-specific indirect object references. For example, instead of using the resource’s database key, the application should map the user indirect reference back to the actual database key on the server.

**Path Traversal**

- **Attacker:** The attacker aims to access files and directories that are stored outside the web root folder. This attack can be executed with an external malicious code injected on the path parameters. For example: An attacker can exploit a file download functionality to download sensitive configuration files, such as the Web.config, and gain vital information about the application such as database credentials, admin credentials, etc.

- **Defender:** Perform proper input validation on all user-supplied parameters and URI requests. Restrict the user-defined path within a whitelist of allowed paths. The directory/filename should be expanded to its absolute canonical path. Enforce directory level access control.

**Forced Browsing**

- **Attacker:** The attacker attempts to access the pages of admin/privileged users. He or she initiates a direct request attack wherein he tries accessing sensitive resources by directly browsing to the URL. For example: An attacker may be able to access administrative pages in www.vulnerableapp.com by browsing to www.vulnerableapp.com/admin.apsx.

- **Defender:** Do not make the assumption that resources can be reached only through the user interface or by the menu items displayed to the user. Enforce authorization at the server side to ensure the user has the required privilege to access the page. Do not rely on client side validation. Perform server side access control check for all pages/functionalities.

**Session Hijacking**

- **Attacker:** The attacker gains access to the active session of an authenticated user and using the session gains full access to all functionalities in the privileges of the
Regenerate session IDs after every successful login and at frequent intervals. Use unique, sufficiently long, random session identifiers to reduce risk of brute force attack.

Attack Tree with Mitigations for Privilege Escalation Functionality-Related Vulnerabilities

- View state should be used to avoid tampering.
- Function level access control should be enabled.
- Whitelist validation should apply to all the user-controlled data. Escape malicious characters in user input.
- Perform server side authorization checks.
- Avoid client side validation.
- Implement secure session management. Use strong session IDs, protect them in transit and regenerate session identifiers at frequent intervals.

- Use a server-generated filename if storing uploaded files on disk.
- Inspect the content of uploaded files, and enforce a whitelist of accepted, non-executable content types.
- Enforce a whitelist of accepted, non-executable file extensions.
- If uploaded files are downloaded by users, supply an accurate non-generic content-type header.
victimized user. If the victim account has elevated privileges, the attacker can even revoke the admin privileges from the victim account and grant it to him- or herself.

- **Defender:** Properly invalidate the session once the user has successfully logged out. Maintain a standard session time out – say, 20 minutes. Do not use static values of session identifiers for the identification of a legitimate user. Do not accept client-supplied session tokens to prevent session fixation. Regenerate session IDs after every successful login and at frequent intervals. Use unique, sufficiently long, random session identifiers to reduce risk of brute force attack. Include http only and secure flags set in cookies in order to avoid session cookie theft.

Figure 8 (see previous page) depicts an attack tree for user privilege management, showcasing attacks that exploit vulnerabilities in the application to gain greater privilege access. Possible remediation methods for preventing privilege escalation and maintaining access control are also presented.
ABOUT THE AUTHOR

Dr. Sivakumar Kathiresan
Principal Architect, Technology
Sivakumar.Kathiresan@cognizant.com

Dr. Sivakumar Kathiresan, B.E., M.E., Ph.D., is a Principal Architect, Technology, within Cognizant’s Enterprise Risk and Security Solutions business unit. In this role, he leads the North American competency, solutions and pre-sales effort in the organization’s integrated vulnerability management services team. He has managed 150-plus security assessment projects across various industry sectors over the last six years. Sivakumar has 22 years of experience, including industry, research and academia, and has delivered more than 125 knowledge-sharing and solution architect sessions on various fields of enterprise security at different forums. His current areas of interest are web security, secure SDLC, advanced log analysis, application vulnerability correlation, integrated vulnerability management, advanced persistent threats and management, and security analytics. Sivakumar received his Ph.D. from the Indian Institute of Technology, Roorkee; he continues to research the area of digital security. His certificates include CEH, CISM, Sourcefire, Qualysguard, Envision, LanDesk and BigData Associate.

ACKNOWLEDGMENTS

The author recognizes the value added by the following Cognizant associates:

• Sivakumar Subramaniam, Head, ITM and IVM.
• Sivarama Subramanian Kailasam and the entire IVM delivery team.

He also acknowledges the contributions to this white paper of the following Cognizant cybersecurity experts:

Balachanthar Palanivelu
Rakesh Balasunder
Subadeepam Rajappan
Nisha Selvaraj
Naga Pranavi K
Padma Prasoona K
Parkavi Neelakandan
Jemmi Angelin Manoharan
Jayasree Otukuru
Abhijeet Ananda Patil
Siddesh N
Rekha Tharmaraj
Pavithra Kamaleswar
Pradeep Varadarajan

Vimalaasree Anandhan
Saravanakumar G
Ranjith Kumar Ramu
Dinesh Jain
Nagamarimuthu Karuppiah
Grace Catherine Jothisprakash
Sivapradha Sivaraman
KK Ashwin
Eby Mohan
Chendhil Thirumalai Kandasamy
Prasath J
Madhan Mohan
Subhashini Sundaramurthy
Rajesh Chilukuri Thirupathaiah
ABOUT COGNIZANT ENTERPRISE RISK AND SECURITY SOLUTIONS

Cognizant Enterprise Risk and Security Solutions (ERSS) business unit specializes in providing end-to-end information security solutions for various industry sectors, including retail, banking and financial services, logistics, telecom, healthcare, manufacturing, and travel and hospitality, and has served 450-plus customers across various geographies. The team has expertise in providing information security solutions and services based on best-of-breed products in each category of enterprise security. Our services include:

- 1600-plus security consultants specializing in IAM, GRC, data security and application security assessment.
- 350-plus CISA, CISM, CISSP, CEH and vendor certified associates.
- 350-plus Infrastructure Security trained associates.
- Over 11000 person years of information security experience.
- A proven track record and experience in 500-plus client engagements for security services.
- Partnership with leading vendors such as IBM, CA, Oracle, Sail Point, Novell, Dell, RSA, HP, Symantec, etc.

ABOUT COGNIZANT

Cognizant (NASDAQ-100: CTSH) is one of the world’s leading professional services companies, transforming clients’ business, operating and technology models for the digital era. Our unique industry-based, consultative approach helps clients envision, build and run more innovative and efficient businesses. Headquartered in the U.S., Cognizant is ranked 230 on the Fortune 500 and is consistently listed among the most admired companies in the world. Learn how Cognizant helps clients lead with digital at [www.cognizant.com](http://www.cognizant.com) or follow us @Cognizant.