This is a post-peer-review, pre-copyedit version of an article published in IEEE Security & Privacy. The final authenticated version is available online at: https://doi.org/10.1109/MSP.2017.13



# Practical Evaluation of Internet Systems' Security Mechanisms

**Paweł Lubomski and Henryk Krawczyk** | Gdańsk University of Technology

**A proposed Internet systems security layer with context-oriented security mechanisms reduces the risk associated with possible vulnerabilities. A metric of the system trust level is proposed and then evaluated according to a university Internet system.**

Internet systems are widespread, accessible via web browsers with the intensive use of Ajax. They often have distributed and service-oriented architectures. Because of their wide and common access by many users, they're exposed to many threats and attacks. These systems are now processing more and more sensitive data—even that of strategic importance for organizations. This means they need high-level security, which is very hard to achieve. Furthermore, current systems are so complicated that they're almost impossible to develop without any mistakes, even when using the best project patterns.

In this article, we propose a practical method for evaluating overall system trust levels based on the well-known standardized audit tests approach (using expert judgment). We show how to organize a system security–hardening process by stepwise improvement of security mechanisms with a new class of such mechanisms, called context-oriented role-based access control (CoRBAC). We've demonstrated on a real system that such approaches are promising in comparison to the standard role-based access control (RBAC) approaches.

## Security Policy

At the beginning of designing such a system, a suitable information security policy should be assumed. Two questions should be considered at that time: How will the system be protected, and what's the acceptable risk level? The aim of such a policy is to protect system authentication, authorization, confidentiality, integrity, nonrepudiation, and availability. The first five areas relate to proper access control; the last one relates to achieving high performance and availability.

The proposed security policy should also account for any possible categories of threats. In the case of Internet systems, the majority of known threats can be divided into two groups on the basis of their impact on the system. The first group of threats is located on the system or software level and is mainly associated with communication between a user and a service. These threats mostly result from technical implementation weaknesses or business logic faults. The other group mostly covers denial-of-service attacks and is guarded mainly on the network control level.

To deal with so many possible threats, a suitable classification and categorization system is required. The best classification in the case of Internet systems' security flaws is the Open Web Application Security Project (OWASP) Top Ten (www.owasp.org/index.php/Category:OWASP_Top_Ten_Project), which describes the 10 most common categories of

weaknesses related to these systems. As of this writing, these are injection, broken authentication and session management, cross-site scripting (XSS), insecure direct object references, security misconfiguration, sensitive data exposure, missing function level access control, cross-site request forgery (CSRF), use of components with known vulnerabilities, and unvalidated redirects and forwards.

Because of the assumed security policy's implementation, various security mechanisms are developed to protect the system. They're implemented in the system because they can't rely on users' web browsers, which aren't under the system administrators' control. The proper design of such complex systems is very burdensome and costly; an undetected system weakness could be considered as a potential threat. To avoid that, periodic security audits can detect vulnerabilities. Figure 1 presents a system security–hardening process, consisting of two cycles. The first cycle represents the improvement process of security mechanisms installed in the system, and the second focuses on periodic verification of system trust. In other words, the system is verified according to whether the desired risk level corresponds to the required security level. The two cycles can be repeated until an acceptable level of system risk is achieved.

The system security–hardening process can consist of a few independent activities. First, there's the removal of the detected vulnerabilities in a specific order (on the basis of their criticality). Next, there are a few ways to lower the risk of new potential vulnerabilities. For example, one step to lowering such risk could be to add additional security mechanisms such as a two-factor authentication. Another step might account for more context parameters. Yet another step might implement better detection mechanisms for irregular behavior, including those that use artificial intelligence algorithms. This process should be done incrementally with a security audit to check whether the change lowers the risk or not. Incremental assessment is important, because adding new security mechanisms can cause a decrease in system usability. This could lead to extreme cases in which system security would decrease because users bypass inconvenient security mechanisms.[1,2]

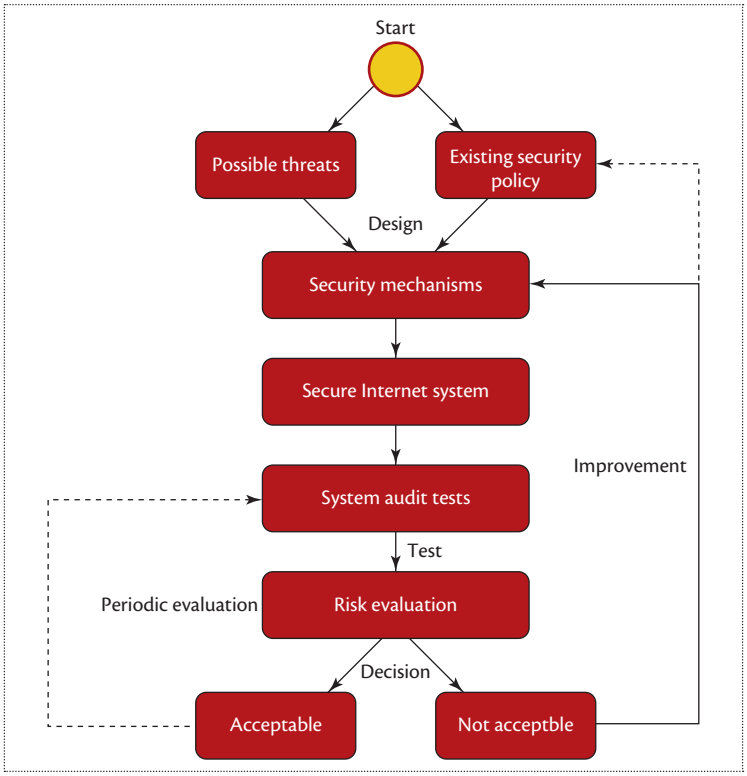


**Figure 1.** System security–hardening process, consisting of two cycles. The first cycle represents the improvement process of security mechanisms installed in the system, and the second focuses on periodic verification of system trust.

## System Security Mechanisms

The traditional security approach is static and insufficient nowadays. The pure RBAC model has been frequently criticized for its inflexibility in rapidly changing domains.[6] (For more information, see the "Role-Based Access Control Approaches" sidebar.) The real-life roles of users in an organization are evaluated very often, whereas RBAC roles are static, and their assignment doesn't depend on other factors, such as time, localization, or other user attributes that change dynamically. Figure 2 presents a two-level user access control to the system with the traditional RBAC security mechanisms, which is the same for regular users and external services. First, an authentication process is performed. Next, permission verification is performed on the basis of permission configuration according to users' connections, their roles, their roles' permissions, and permissions guarding the services.

Users interact with the system through an interface and environment that are described by many parameters that define the context of their activities.[5–8] The context might be a certain period of time in which the user action is performed (weekday, Saturday, or Sunday). It can be a user's physical (US, Poland, Germany, UK, or France) or logical localization (internal network, campus network, or Internet).[9] It can correspond to the general state of the system (regular work, under high load, or under maintenance) or to the relation between the user and the available data (a physician and medical records of his or her patient).[10] In addition, the user's interaction history (log of previous actions in the system) or the kind of device being used (standard PC or mobile device) can be taken into account.[11] The context information can be gathered from many different sources, such as user requests made by a web

## Role-Based Access Control Approaches

The role-based access control (RBAC) approaches are still under improvement to satisfy requirements related to mobile and cloud computing. First, context expansion is considered to create dynamic and flexible access control mechanisms. To satisfy large numbers of users with different demands, extra attributes have been added to the approach. This has led to a new approach, called attribute-based access control (ABAC).[1] The ABAC model is a generalized version of the RBAC model, where access rights are granted to users using policies that combine multiple attributes. It's important to note that such attributes relate not only to relationships between users and the available data describing their profiles but also to other types of entities that represent users and systems, such as hardware and software configuration, communication characteristics, and even security policy.[2] Proper assignments of attribute values to different entities are necessary to protect against unauthorized access. The problem lies in how to estimate the suitable number of attributes and how to manage them to achieve the required level of credibility. (Some propositions have been made for cloud tenants.[3])

Our proposed context-oriented role-based access control (CoRBAC) model is an exact solution to such problems and corresponds to the direction of access control development mentioned earlier. It provides precise and effective security mechanisms, working on the basis of historical and current contextual information collected from the system and environment. The proposed solution suits any service-oriented Internet system, regardless of changes in the environment, because authorization is made on the system side and only context parameters that are accounted for are changed.


**References**

1. D.R. Kuhn, E.J. Coyne, and T.R. Weil, "Adding Attributes to Role-Based Access Control," *Computer*, vol. 43, no. 6, 2010, pp. 79–81.
2. Q.M. Rajpoot, C.D. Jensen, and R. Krishnan, "Integrating Attributes into Role-Based Access Control," *Proc. Data and Applications Security and Privacy XXIX*, 2015, pp. 242–249.
3. T. Mather, S. Kumaraswamy, and S. Latif, *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*, O'Reilly, 2009.


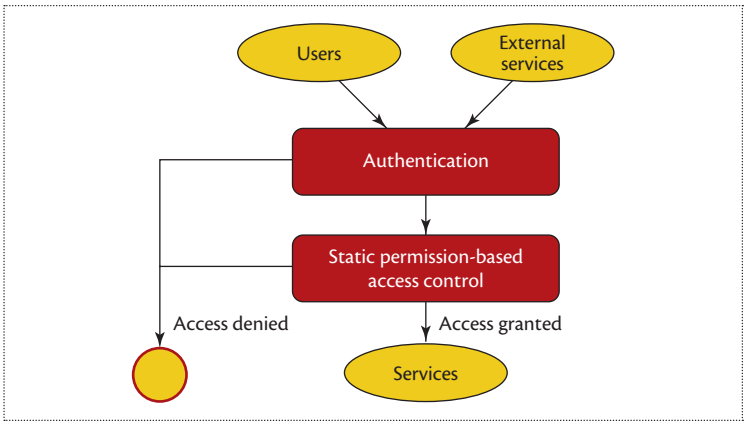


**Figure 2.** Traditional role-based access control (RBAC) security mechanisms. First, an authentication process is performed. Next, permission verification is performed on the basis of permission configuration according to users' connections, their roles, their roles' permissions, and permissions guarding the services.

browser, system information status, and a system clock. The ability to integrate contextual information makes the role-based security model flexible, thus minimizing the risk of potential threat. In this way, we achieve dynamic security mechanisms that can adapt to the changing security policies during runtime with minimal loss of functionality and with little or no manual assistance. They do a good job of addressing events such as changes in personnel, changes in the execution environment, and crisis situations.

Let's consider the CoRBAC model, which is an extension of the RBAC model, where widely understood context is taken into account. Each context parameter comprises a finite, discrete set of possible values. Some context parameters have a continuous character (for example, time). During context analysis, they're clustered into certain groups, such as days of the week. The same situation applies to context parameters with discrete numerical values, such as a set of IP addresses. They're clustered into subsets of a specific netmask. A vector of the current values of each context parameter describes the current context accompanying each user interaction with the system. Each context parameter is gathered separately. This approach lets us implement specialized mechanisms to gain different context parameters in parallel.

Figure 3 illustrates the CoRBAC model's access control activities. These consists of two authentication activities: a basic authentication (the same as in the RBAC model) and a user trust–based access control that has extra security mechanisms dependent on the system's trust in the user. (One possible solution is described in detail in the 2015 proceedings of the Computer Network Conference.[12])

The user trust analysis and access control account for the user profile (history of contexts accompanying the user with the system interaction) and the current context. It lets us detect irregular behavior. Thus, the system security layer learns users' behavior. Each

user profile contains context values clustered into groups by how often the user is in a particular context. This can be done, for example, using the hierarchical agglomerative clustering algorithm, a "bottom up" approach in which clusters are merged in pairs and moved up in the hierarchy. This way, we can achieve dendrogram—an extensive hierarchy of clusters that merge with each other at certain distances.[13] The resulting groups are translated into trust levels. On the basis of the computed trust level, the appropriate security mechanism is fired. Some examples of such extra security mechanisms are reentering a password or other personal data, inputting a one-time-password sent by SMS, and selecting a previously chosen image from a list. If the user satisfies this mechanism, his or her identity is confirmed.

In dynamic permission-based access control (see Figure 3), first, the appropriate permissions are assigned to the user on the basis of the configuration (connections among users, roles, and permissions). This is done exactly as in the traditional RBAC model. Next, the permission set is limited, based on permissions' association with some context values. This step applies only to the CoRBAC model. In this way, in any particular situation, the current context dynamically determines the "active (allowed) permissions" set, which is a subset of the "all permissions" set specified in the RBAC model. This process is very important in risk analysis—it reduces the scope of impact of a potential security incident. After that, the permission verification takes place on the basis of access rules. If this verification is met, the requested access to services is granted or denied.

According to the CoRBAC model, these two levels of access control can be represented by the following procedures, written in pseudocode:

```
userTrustBasedAccessControl (user,
service):

    1. currentContext =
        getCurrentContext()
    2. userProfile =
        getUserProfile(user)
    3. trustLevel = computeUser
       TrustLevel(userProfile,
       currentContext)
    4. result = performUserSecurity
        Check(trustLevel)
    5. if(result == true)
       then call
       permissionBasedAccessControl
        (user, service)
       else return DENY_ACCESS
```

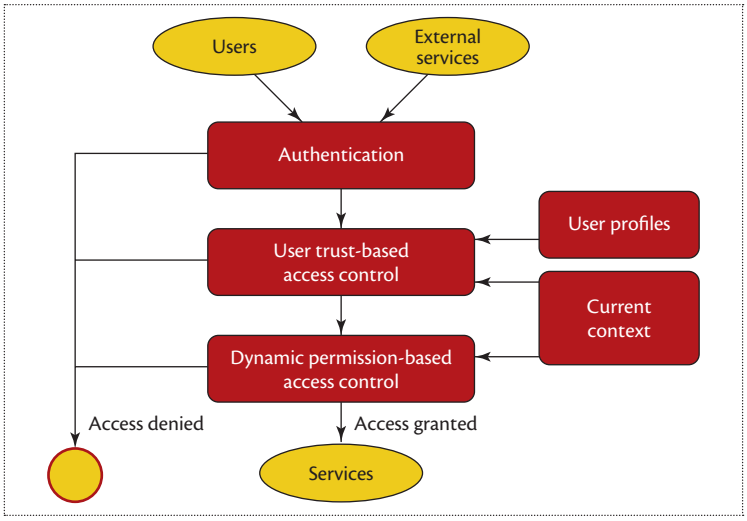


**Figure 3.** Use of the proposed context-oriented role-based access control (CoRBAC) security mechanisms. Compared to a traditional RBAC security mechanisms (see Figure 2), there's an added user trust-based access control that has extra security mechanisms dependent on the system's trust in the user.

```
permissionBasedAccessControl (user,
service):

    1. roles = getRolesOfUser(user)
    2. userPermissions =
        getPermissionsOfRoles(roles)
    3. currentContext =
        getCurrentContext()
    4. contextPermissions = getContext
        Permissions(currentContext)
    5. userPermissions =
        userPermissions ∩
        contextPermissions
    6. servicePermissions =
        getServicePermissions(service)
    7. if(userPermissions ∩
        servicePermissions != ∅)
        then return GRANT_ACCESS
        else return DENY_ACCESS
```

where ∩ indicates the intersection of the sets. For comparison, the RBAC model includes only four of the above steps (from lines 1, 2, 6, and 7).

## Proposed System Risk and Trust Evaluation Method

It's difficult to correctly determine a system's level of security. The key challenges are to measure the lack of incidents and choose the proper representative metrics. Fortunately, many open initiatives address these problems. For Internet systems on the web application level,

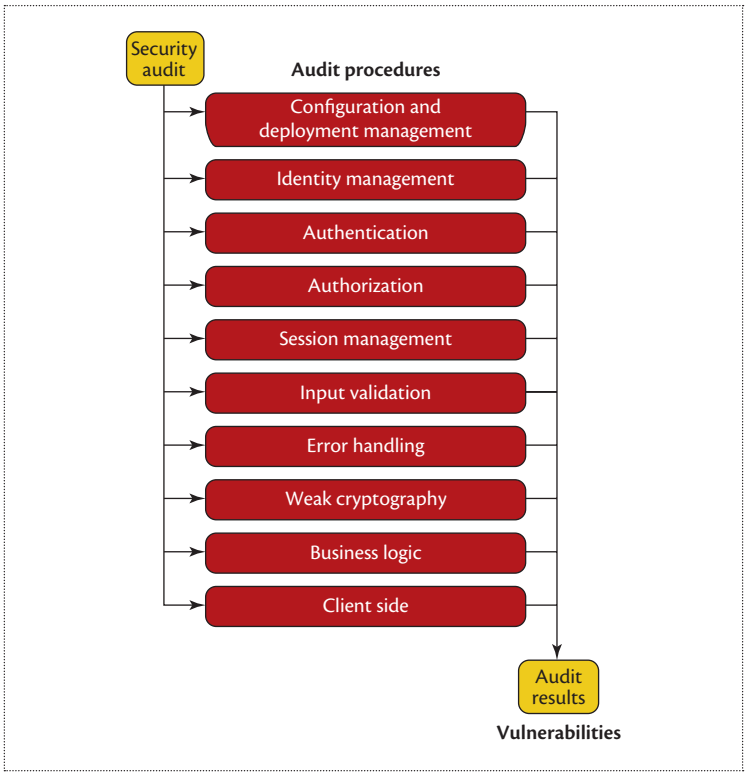


**Figure 4.** The scope of the standard audit procedure in accordance with Open Web Application Security Project Testing Guide. It consists of 10 key areas that are analyzed during such an audit.

the best approach currently seems to be OWASP (www.owasp.org). It gathers data from many specialists and scientists in the field of web application security to classify threats, define techniques and guides, and indicate how to make systems more secure and how to test systems. The biggest advantage of the results is that they're widely available via open licenses (such as Creative Commons; creativecommons.org) and can be used for different kinds of systems.

One possible way to measure the level of a system's security is using security audits, which are mostly carried out by external auditors who check the system for known potential security vulnerabilities listed in the OWASP Top Ten or Common Weakness Enumeration (CWE; cwe.mitre.org). Moreover, the technical security audit can be done on the basis of the OWASP Testing Guide (www.owasp.org/index.php/OWASP_Testing_Project). Figure 4 presents the key areas of such an audit. As a result, the detected vulnerabilities are reported to the system owners and can be submitted to Common Vulnerabilities Enumeration (CVE; cve.mitre.org).

Each vulnerability detected during the audit should be the subject of risk analysis to determine the level of the system risk and, consequently, how the potential security incident associated with that vulnerability impacts an organization's functionality.[14] On the basis of the list of detected vulnerabilities (which should be updated periodically), prioritization of remedial work should be done; vulnerabilities with highest criticality should be carried out first.[15]

Threats can be categorized into the OWASP Top Ten categories. Other classifications are available (such as CWE), but it seems that the OWASP Top Ten covers most of the threat categories corresponding to Internet systems. In practice, in complex and advanced systems, only rarely can we discover vulnerabilities of other categories. In those cases, after the analysis and acceptance, the OWASP Top Ten can either cover it, or some actualization of the classification can be performed. Each detected vulnerability should be categorized and ranked by a suitable risk metric.

There are a few methods and techniques for risk analysis and metrics evaluation. The most common is the Common Vulnerability Scoring System (CVSS; www.first.org/cvss). The CVSS is especially convenient because NIST has published a CVSS special online calculator (nvd.nist.gov/CVSS/v2-calculator) for such analysis.

Another approach, STRIDE,[16] is associated with the Microsoft DREAD threat-risk ranking model.[17] However, for Internet systems, it seems that the OWASP Top Ten/CWE with CVSS is more suitable for such analysis, as they better address the web systems' aspects.

At the time of this writing, we're using version 2 of the CVSS, which consists of three characteristic groups:

- *base*, the fundamental characteristics of a vulnerability that are constant over time and user environments;
- *temporal*, the characteristics of a vulnerability that change over time but not across user environments; and
- *environmental*, the characteristics of a vulnerability that are relevant and unique to a particular user's environment.

There are six base characteristics that refer to each detected vulnerability: access vector, access complexity, authentication, confidentiality impact, integrity impact, and availability impact. The first three metrics capture how the vulnerability is accessed and whether extra conditions are required to exploit it. The latter three metrics measure how a vulnerability, if exploited, will directly affect an IT asset, where impacts are independently defined as the degree of loss of confidentiality, integrity, and availability. For example, vulnerability might cause a partial loss of integrity and availability but no loss of confidentiality.

Two other characteristic groups, which are optional, are very specific to the organization where the Internet system works; these might change over time during the life of the vulnerability. These two groups are omitted

**Table 1. Correspondence of risk score, risk level, and system trust level (STL).**

| Risk score | Criticality (risk level) | STL impact factor |
|---|---|---|
| 0 | Zero | 1.0 |
| 0–4.0 | Low | 0.6 |
| 4.0–7.0 | Medium | 0.3 |
| 7.0–10.0 | High | 0.1 |

**Table 2. Detected vulnerabilities in GUT Instinct system and their risk score.***

| | | System with role-based access control (RBAC) | | | System with context-oriented role-based access control (CoRBAC) | | |
|---|---|---|---|---|---|---|---|
| ID | Detected vulnerability | CVSS v2 vector | Score** | | CVSS v2 vector | Score** | |
| v1 | Cross-site scripting—vulnerability 1 | (AV:N/AC:M/Au:S/C:C/I:C/A:N) | 7.9 | H | (AV:N/AC:M/Au:S/C:P/I:P/A:N) | 4.9 | M |
| v2 | Cross-site scripting—vulnerability 2 | (AV:N/AC:L/Au:S/C:P/I:P/A:N) | 5.5 | M | (AV:A/AC:L/Au:S/C:P/I:P/A:N) | 4.1 | M |
| v3 | Cross-site scripting—vulnerability 3 | (AV:N/AC:L/Au:N/C:P/I:P/A:N) | 6.4 | M | (AV:N/AC:L/Au:N/C:N/I:P/A:N) | 5.0 | M |
| v4 | Cross-site scripting—vulnerability 4 | (AV:N/AC:M/Au:S/C:C/I:C/A:N) | 7.9 | H | (AV:A/AC:M/Au:S/C:P/I:P/A:N) | 3.8 | L |
| v5 | SQL injection—vulnerability 1 | (AV:N/AC:H/Au:S/C:C/I:C/A:C) | 7.1 | H | (AV:A/AC:H/Au:S/C:C/I:C/A:C) | 6.5 | M |
| v6 | SQL injection—vulnerability 2 | (AV:N/AC:H/Au:S/C:C/I:C/A:C) | 7.1 | H | (AV:A/AC:H/Au:S/C:C/I:C/A:C) | 6.5 | M |
| v7 | SQL injection—vulnerability 3 | (AV:N/AC:H/Au:S/C:C/I:C/A:C) | 7.1 | H | (AV:A/AC:H/Au:S/C:C/I:C/A:C) | 6.5 | M |
| v8 | Cross-site request forgery (CSRF)—vulnerability 1 | (AV:N/AC:M/Au:S/C:N/I:P/A:N) | 3.5 | L | (AV:N/AC:M/Au:S/C:N/I:P/A:N) | 3.5 | L |
| v9 | CSRF—vulnerability 2 | (AV:N/AC:M/Au:S/C:N/I:C/A:N) | 6.3 | M | (AV:A/AC:M/Au:S/C:N/I:P/A:N) | 2.3 | L |
| v10 | Session ID—vulnerability 1 | (AV:N/AC:H/Au:S/C:C/I:C/A:N) | 6.6 | M | (AV:N/AC:H/Au:S/C:P/I:P/A:N) | 3.6 | L |
| v11 | Session ID—vulnerability 2 | (AV:N/AC:H/Au:S/C:C/I:C/A:N) | 6.6 | M | (AV:A/AC:H/Au:S/C:C/I:C/A:N) | 5.9 | M |
| v12 | Password reset procedure | (AV:N/AC:L/Au:N/C:P/I:N/A:N) | 5.0 | M | (AV:N/AC:L/Au:N/C:P/I:N/A:N) | 5.0 | M |

**Access vector (AV) can be local (L), adjacent network (A), or network (N). Access complexity (AC) can be high (H), medium (M), or low (L). Authentication (Au) can be multiple (M), single (S), or none (N). Confidentiality impact (C) can be none (N), partial (P), or complete (C). Integrity impact (I) can be none (N), partial (P), or complete (C). Availability impact (A) can be none (N), partial (P), or complete (C).*

*** The scores are marked using three colors that correspond to low, medium, and high levels of risk—respectively, green, yellow, and red.*

in our considerations because we want our system trust level assessment for the considered two cases (the system with RBAC and CoRBAC security mechanisms) to be independent of organizational conditions and take into account the Internet system rather than the specific organization where it's in place.

CVSS v2 defines three discrete metric values for each characteristic. For each of the base characteristics, the metric values are as follows:

- access vector (AV) can be local (L), adjacent network (A), or network (N);
- access complexity (AC) can be high (H), medium (M), or low (L);
- authentication (Au) can be multiple (M), single (S), or none (N);
- confidentiality impact (C) can be none (N), partial (P), or complete (C);
- integrity impact (I) can be none (N), partial (P), or complete (C); and
- availability impact (A) can be none (N), partial (P), or complete (C).

On the basis of these metrics, it's possible to determine a numerical score that reflects the vulnerability's criticality (the higher the risk score, the higher the criticality). Then, it's possible to compare it to the other detected vulnerabilities. CVSS v2 defines three ranges

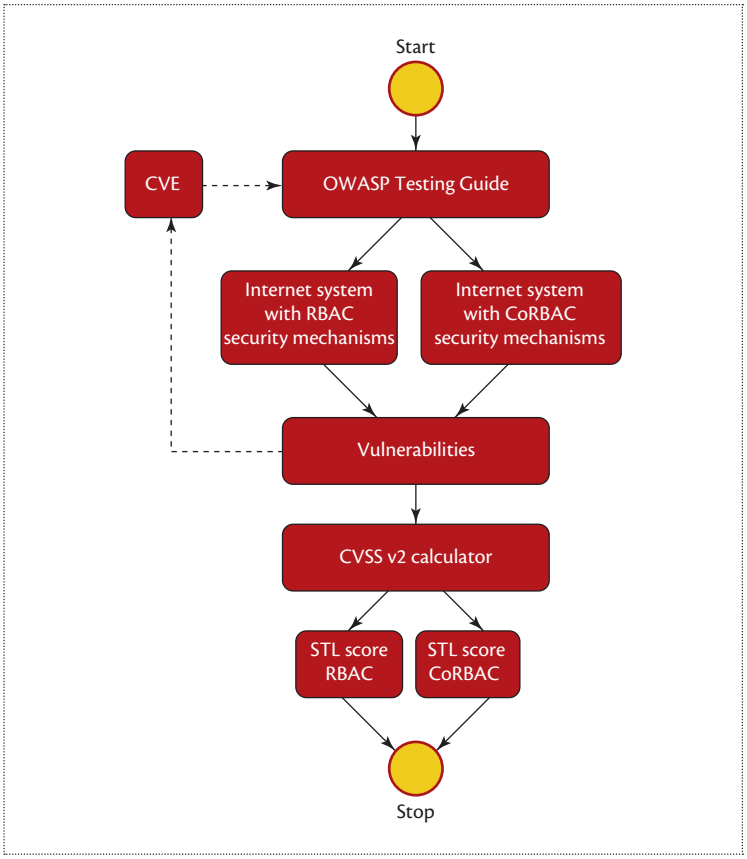


**Figure 5.** System trust level (STL) score evaluation for the GUT Instinct system with RBAC and CoRBAC security mechanisms (one iteration of Figure 1). The STL for the system with these two variants of security mechanisms is calculated separately. The upper part depicts the audit process, where the newly detected vulnerabilities are reported to the Common Vulnerabilities Enumeration (CVE). The lower part shows the process of vulnerabilities criticality (Common Vulnerability Scoring System [CVSS] calculator) as well as STL score calculation.

of criticality: low (score of 0–4.0), medium (score of 4.0–7.0), and high (score of 7.0–10.0). For each system, we can evaluate the number of vulnerabilities with zero, low, medium, and high risk level—that is, number Z, number L, number M, and number H, respectively. Table 1 presents mapping across the risk score, criticality (according to CVSS v2), and the proposed system trust level impact factor.

The overall system trust level (STL) is defined as

$$STL = \frac{nZ + nL + 0.6 + nM + 0.3 + nH + 0.1}{nT},$$

where

- $nZ$ represents the number of potential vulnerabilities that were checked that don't exist in the system;
- $nL$, $nM$, and $nH$ represent the number of vulnerabilities of criticality L, M, and H detected during the audit, respectively; and
- $nT$ represents the total number of vulnerabilities checked during the audit.

The minimum possible value of system trust is 0.1 and the maximum value is 1.0. In practice, this value should be almost 1.0.

## The Case Study Analysis

GUT Instinct is an Internet system that was implemented and has been used at Gdańsk University of Technology (GUT). It provides students, teachers, and other employees, as well as external cooperating individuals including entrepreneurs (about 40,000 active users), with many functional services. GUT Instinct as the central platform of the university systems gathers data and supports processes in the most important areas of the organization's activity, such as education, research, innovation, and cooperation.

This system regularly undergoes security audits, based on the methods described in the previous section, which are performed by an external, certified auditing company. Each audit consists of penetration tests and an IT infrastructure check. Experts perform both automated penetration testing and manual tests of application and configuration. Each potential vulnerability detected by an automatic scan is verified and analyzed on a deeper level by an auditor. An infrastructure overview also occurs. Optionally, there can be a code review and some social engineering tests carried out on the basis of the OWASP Top Ten, the OWASP Testing Guide, and the auditors' experience (see Figure 5). Each system functionality is checked against all types of vulnerabilities.

More than 1,000 potential vulnerabilities are tested during each audit. Table 2 presents only the detected vulnerabilities during the first security audit of the GUT Instinct system. For each vulnerability found in the system, a risk score from the NIST CVSS v2 calculator was assigned. The scores are marked using three colors that correspond to low, medium, and high levels of risk—respectively, green, yellow, and red. As shown in Figure 5, we compared the system's STL score using the traditional security model (RBAC), followed by the context-oriented model (CoRBAC). Table 3 summarizes a comparison of both solutions, giving the numbers of vulnerabilities obtained for each criticality (risk level) as well as the final STL scores.

The overall STL (according to our earlier definition) is 0.9909 for the system using RBAC and 0.9928 for CoRBAC. (Remember that values of high reliability and dependability system metrics are close to

**Table 3. Number of vulnerabilities of each criticality detected in the system using RBAC and CoRBAC security mechanisms.**

| Criticality (risk level) | Zero | Low | Medium | High | STL score |
|---|---|---|---|---|---|
| No. vulnerabilities—RBAC | 993 | 1 | 6 | 5 | 0.9909 |
| No. vulnerabilities—CoRBAC | 993 | 4 | 8 | 0 | 0.9928 |

0.99999.) It's clearly noticeable that the system with the context-oriented security has a higher level of system trust.

The weakest point in our approach is that it relies, to some extent, on an auditor's subjective evaluation. Thus, it depends on the knowledge and experience of the expert who performs exhaustive tests, with additional support of some advanced tools. But it still seems to be the best way to measure the system security level.

User profiles, built on behaviors when using the system, seem to be the most promising area of context analysis. Our proposed access control mechanisms based on user trust needs some improvement in two areas: an extension of the complexity of the analyzed context, and an improved detection of false-positive and false-negative cases.

Although CVSS v2 characteristics correspond to two types of context (user localization and limitation of user access to various system data), in practice, some other previously mentioned context parameters can be taken into account. Consequently, the set of CVSS v2 metrics was slightly modified in CVSS v3, which considers more dynamic and contextual aspects. So, future work will be based on this version.

The hardening of system security is an incremental process (compare Figure 1 and Figure 5). In each iteration, the security layer is improved. Consequently, after several iterations, we might approach a 0.99999 system trust level. However, it's essential for new security mechanisms not to decrease the user convenience significantly. It works much better when the system security layer is almost unnoticeable to the user. We hope to address this area in future research.

## References


1. S.P.S. Pahnila, M.S.M. Siponen, and A.M.A. Mahmood, "Employees' Behavior towards IS Security Policy Compliance," *Proc. 40th Ann. Hawaii Int'l Conf. System Sciences* (HICSS 07), 2007; doi:10.1109/HICSS.2007.206.
2. S. Furnell, "Usability versus Complexity—Striking the Balance in End-User Security," *Network Security*, vol. 2010, no. 12, 2010, pp. 13–17.
3. J.M. Stanton et al., "Analysis of End User Security Behaviors," *Computers and Security*, vol. 24, no. 2, 2005, pp. 124–133.
4. M. Strembeck and G. Neumann, "An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments," *ACM Trans. Information and System Security*, vol. 7, no. 3, 2004, pp. 392–427.
5. F. Cuppens and N. Cuppens-Boulahia, "Modeling Contextual Security Policies," *Int'l J. Information Security*, vol. 7, no. 4, 2007, pp. 285–305.
6. X. Jin, R. Krishnan, and R. Sandhu, "A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC," LNCS 7371, 2012, pp. 41–55; doi:10.1007/978-3-642-31540-4_4.
7. Z. Maamar, D. Benslimane, and N.C. Narendra, "What Can Context Do for Web Services?," *Comm. ACM*, vol. 49, no. 12, 2006, pp. 98–103.
8. R. Mayrhofer, H.R. Schmidtke, and S. Sigg, "Security and Trust in Context-Aware Applications," *Personal and Ubiquitous Computing*, Nov. 2012; doi:10.1007/s00779-012-0630-2.
9. M.L. Damiani et al., "GEO-RBAC," *ACM Trans. Information and System Security*, vol. 10, no. 1, 2007, article 2.
10. L. Sliman, F. Biennier, and Y. Badr, "A Security Policy Framework for Context-Aware and User Preferences in E-Services," *J. Systems Architecture*, vol. 55, 2009, pp. 275–288.
11. A. Gupta, M.S. Kirkpatrick, and E. Bertino, "A Formal Proximity Model for RBAC Systems," *Computers and Security*, Sept. 2013; doi:10.1016/j.cose.2013.08.012.
12. H. Krawczyk and P. Lubomski, "User Trust Levels and Their Impact on System Security and Usability" *Comm. Computer and Information Science*, vol. 522, 2015, pp. 82–91.
13. A. Bouguettaya et al., "Efficient Agglomerative Hierarchical Clustering," *Expert Systems with Applications*, vol. 42, no. 5, 2015, pp. 2785–2797.
14. N. Dimmock et al., "Using Trust and Risk in Role-Based Access Control Policies," *Proc. 9th ACM Symp. Access Control Models and Technologies* (SACMAT 04), 2004, pp. 156–162.
15. P. Damián-Reyes, J. Favela, and J. Contreras-Castillo, "Uncertainty Management in Context-Aware Applications: Increasing Usability and User Trust," *Wireless Personal Comm.*, vol. 56, no. 1, 2009, pp. 37–53.
16. S. Hernan et al., "Uncover Security Design Flaws Using the STRIDE Approach," *Microsoft MSDN Magazine*, 2006; download.microsoft.com/download/3/a/7/3a7fa450-1f33-41f7-9e6d-3aa95b5a6aea/MSDNMagazineNovember2006en-us.chm.
17. J.D. Meier et al., "Improving Web Application Security: Threats and Countermeasures," *Microsoft*

*Patterns and Practices*, 2015; msdn.microsoft.com/en-us/library/aa302419.aspx.

**Paweł Lubomski** is the director of the IT Services Centre at the Gdańsk University of Technology. His research interests include access control, security in context-aware systems, security of large-scale distributed e-service systems, and building secure IT architectures. Lubomski received a PhD in computer science from the Gdańsk University of Technology. Contact him at lubomski@pg.gda.pl.

**Henryk Krawczyk** is a computer science professor in the Faculty of Electronics, Telecommunication and Informatics at the Gdańsk University of Technology. His research interests include distributed processing and Internet systems. Krawczyk received a PhD and an advanced PhD in computer engineering and software engineering, respectively, from the Gdańsk University of Technology. He's a member of IEEE and the Polish Academy of Sciences. Contact him at hkrawk@eti.pg.gda.pl.