

YESTERDAY AND TODAY – THE TRANSFORMATION OF SECURITY SOLUTIONS

A whole new generation of web applications is establishing itself with the success of JavaScript and HTML5. Applications are being networked across organizations at the same time. What does that mean for proven security concepts?

by Stephan Schweizer and Thomas Zweifel

In the classic Internet age at the end of the 1990s and early 2000s the fronts were clearly defined: The “bad” populated the Internet, the “good” did their work on the intranet. What is known as a network perimeter was and is being set up to separate these two worlds. This usually consists of an external and an internal firewall. Additional security components such as secure reverse proxies and web application firewalls are located in the zone in between them, known as the “demilitarized zone” (DMZ). They ensure that only users with strong authentication (through what is known as two-factor authentication) can access the available web applications from the “evil empire”. This barrier of security components also prevents direct communication between the client and application. In addition to single sign-on functionality, such a setup also offers flexible options for intervention in order to protect applications against direct attacks such as denial-of-service (DoS), cross-site scripting and injection attacks (see box).

**AROUND THE TURN OF THE MILLENNIUM,
THE FRONTS BETWEEN GOOD AND
EVIL ON THE INTERNET WERE
STILL CLEARLY DEFINED.**

The application logic is shifting to the client

In this world there were no doubts: The application logic lies on the server side. Providing visual access to content for the user was the primary task of the browser. There was a broad consensus that this was the right approach after the experiences made in the client-server era of the 1990s. Users were very frugal as well, being satisfied with spartan user interfaces (GUIs) that appear archaic from today's perspective.

Denial-of-service (DoS) attacks *are aimed at making the use of a service by regular users impossible. The attacker attempts to overwhelm the service with a large number of requests, or to make it crash by purposefully exploiting known vulnerabilities.*

With distributed denial-of-service (DDoS) attacks, *a large number of Internet clients distributed around the world is used to overload the target system with requests. These clients are usually workstations that were previously infected with a Trojan virus. Subsequently, the system is remotely controlled by the attackers so the owners usually participate in the denial-of-service attack without their knowledge. The large number of sometimes varying clients makes it more difficult for the service provider to filter the DoS requests.*

Cross-site scripting *aims to infiltrate a web application with the attacker's JavaScript code. The attack usually takes place via insufficiently protected input fields of the application or URL query parameters. Depending on the attacker's intentions, the code that is smuggled in uses the browser's runtime environment to intervene in the application logic, load additional malicious code, steal digital identities or exploit existing security vulnerabilities on the client system, for example to install a Trojan virus.*

SQL injection attacks *are aimed at spying out, stealing or modifying data (such as credit card information) stored in application databases. They attempt to inject and execute the attacker's SQL statements in the application through unprotected input fields or query parameters. SQL Injection attacks can be prevented through corresponding WAF filters or on the database side also through “Prepared Statements” (with bound parameters).*

GUI requirements became more elaborate with the development of superior applications. Initial attempts aimed at making GUIs user-friendly were made with JavaScript. At that time however, it would not have occurred to anyone to implement business logic in JavaScript – the elements of an application realized in JavaScript were limited to logic related to visualization. There was good reason why developers were reluctant to implement functionality in the browser: In the former HTML4 world, the tags may have been standardized but the semantics and representation were interpreted very differently by various browser providers. This meant that Web content looked entirely different depending on the browser that was used. The JavaScript API (Application Programming Interface) provided by the browsers was very spartan as well. Those who wanted to tackle more sophisticated tasks with JavaScript while simultaneously gaining control over the diversity of browsers truly had to work hard.

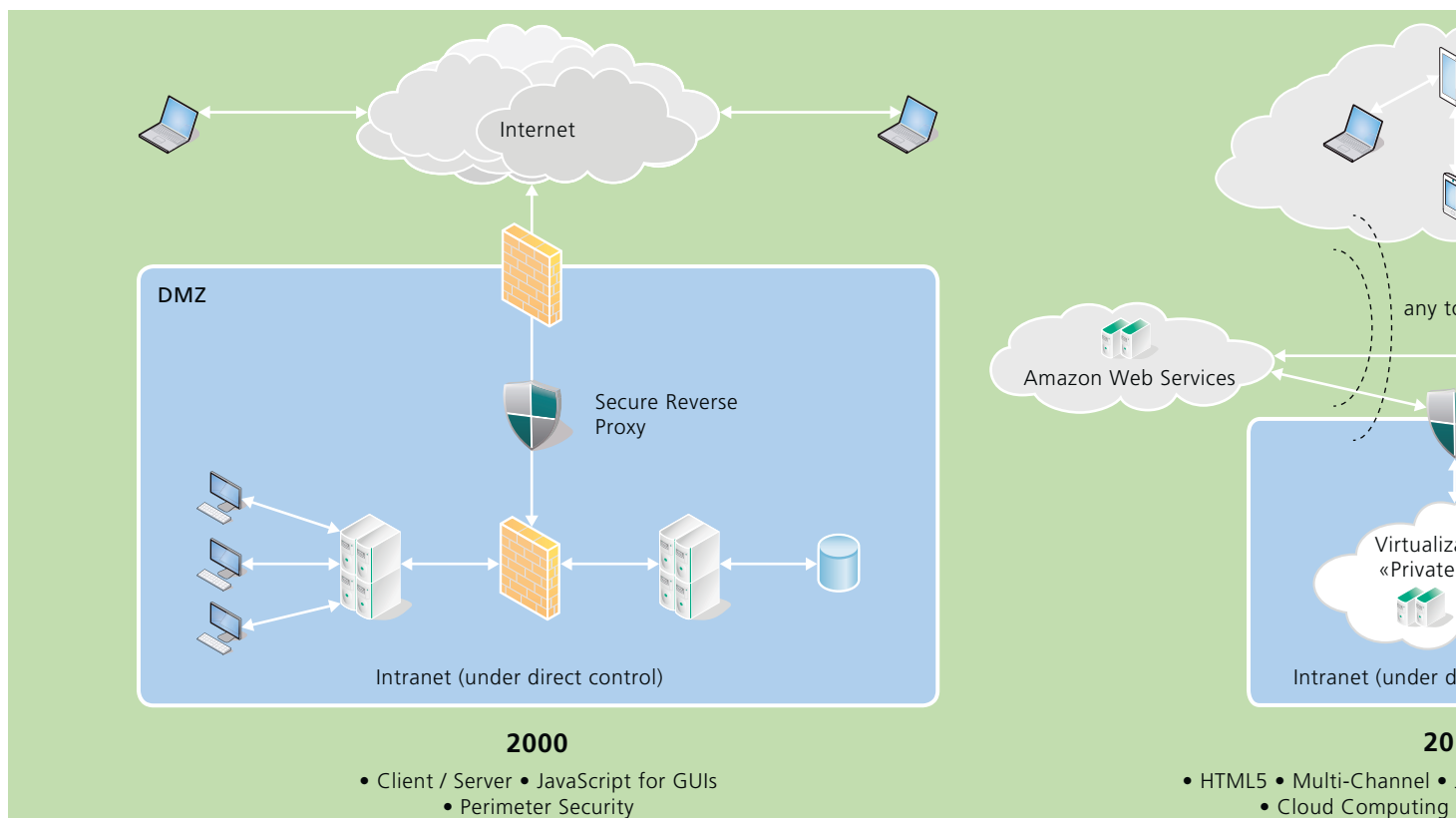
HTML5 rings in a new era

This changed fundamentally with the introduction of HTML5. In addition to unambiguous semantics and uniform representation, HTML5 also provides a very comprehensive and standardized JavaScript API. This reduced the browser-specific differences on the one hand and, on the other hand, a few lines of JavaScript were suddenly enough to skip ahead or back in videos,

draw diagrams, store data locally in the browser using WebStorage, access the camera of a mobile device or use it to determine the geographical location. This paved the way for Rich Internet Applications (RIAs) with comprehensive, purely client-side functions. The browser mutated from a “visualization vehicle” to a comprehensive runtime environment for JavaScript-based applications in this process.

HTML5 OPENED UP ENTIRELY NEW PERSPECTIVES FOR DEVELOPERS AS WELL AS FOR ATTACKERS.

The diverse functions of HTML5 opened up entirely new perspectives for application developers – but unfortunately the same applies to attackers. Even though security aspects were taken into account in the standardization of HTML5: The scope of functionality in the browser alone means that implementation errors result in security vulnerabilities that are relatively easy to exploit. A relatively quiet paradigm shift is currently underway as well: The application functionality is increasingly shifting to the client, which is precisely the weakest link in the entire security chain.



Quo vadis, Internet: Due to the rapid technological development, security needs to be questioned constantly.

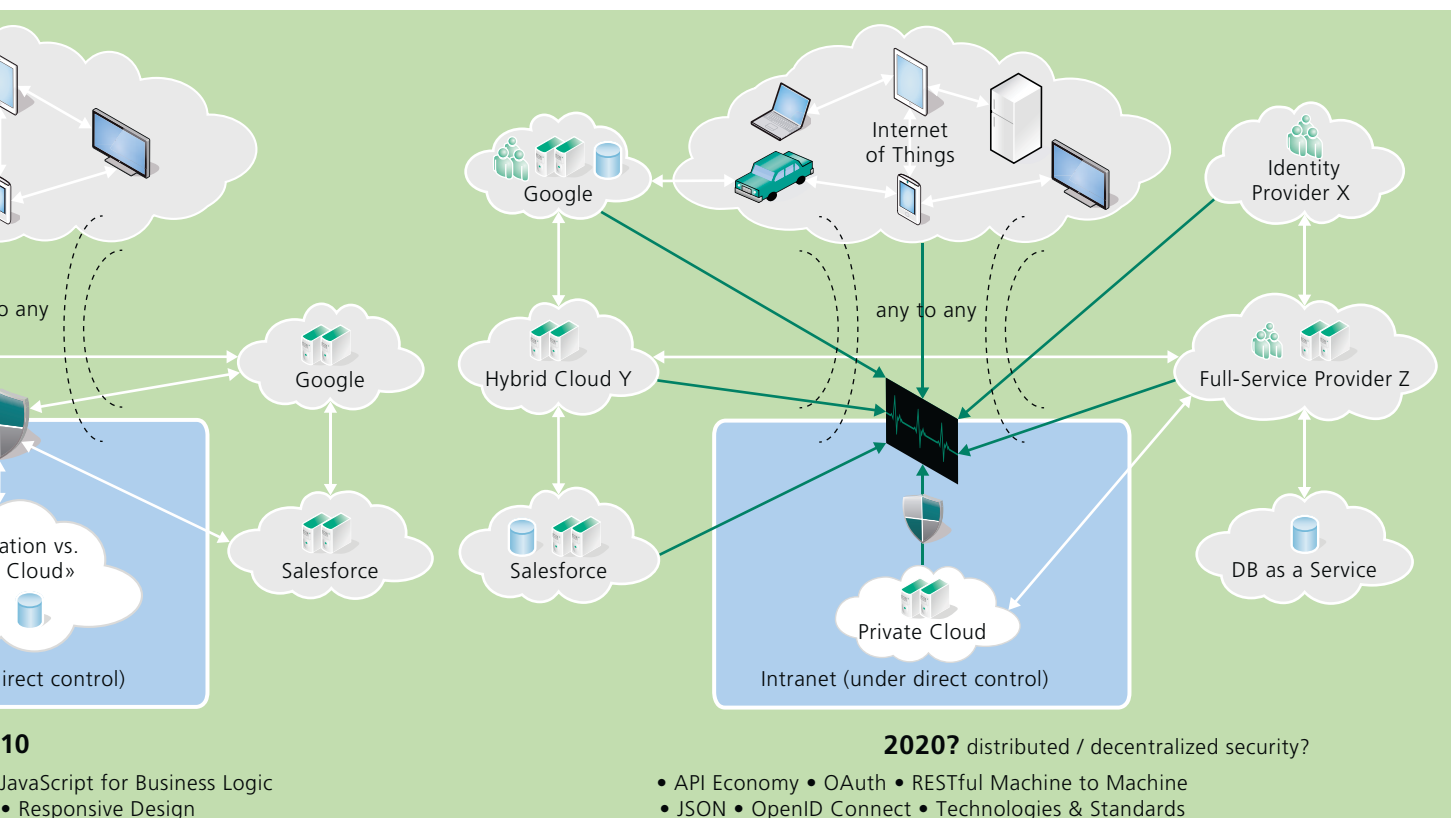
ONCE EXTERNAL CODE HAS BEEN SMUGGLED IN, MANY SECURITY MECHANISMS FAIL.

Because of this shift to the weakest link, some established server-side security mechanisms are no longer effective so that protection is reduced or – in extreme cases – eliminated entirely. This is due to the functionality of modern JavaScript frameworks: Due to the client-side logic, server-side security components have only a limited view of the content effectively displayed by the application.

When an attacker succeeds in smuggling its own JavaScript code into these processes, thereby gaining control, things get challenging: Once external code has been smuggled in, many of the internal browser security mechanisms fail since the malicious JavaScript fragments are viewed as a regular part of the application. This makes the comprehensive HTML5 JavaScript API available to the attacker, for example making it possible to operate the camera, access the HTML5 WebStorage content or use Web Workers (JavaScript background processes) to launch a DoS attack.

Security is more central than ever

The examples emphasize that the consistent protection of web applications is more important today than ever before. However, new concepts are required in order to maintain the past level of security. An integrated security concept has to adequately take the client side into account. In addition to clear coding directives for developers, this also includes the consistent use of the existing security mechanisms offered by modern browsers. While these can be centrally activated and controlled by server-side security components (such as nevisProxy), enforcing the security policies remains on the client side. This means a Security Officer has to trust that the browsers apply the specified policies correctly. It is just as or actually more important to consistently apply the protective mechanisms against injection attacks available on the server. Even though these mechanisms no longer offer the same protection in certain scenarios as they did in the past, they constitute an obstacle in the attempt to smuggle malicious code into an application which is not to be underestimated. All security measures jointly pursue one goal: Preventing the infiltration of malicious JavaScript code under all circumstances.





Stephan Schweizer and Thomas Zweifel: Developing strategies for dealing with new Internet hazards.

New security concepts are in demand

Yet one central question remains unanswered by all the preceding measures: How can an organization prepare for the worst case, that is the success of an attacker in the attempt to smuggle its own code into the application? New solutions definitely need to be developed for this scenario. It can be assumed that an infected JavaScript application will behave differently than the original – which is why server-side anomaly detection is a promising approach. The fundamental idea is that there is server-side intelligence which is able to learn and model the normal behavior of an application. Based on this assumption, it is possible to identify deviating behavior and respond to that. The responses to such a situation depend on the respective context. Conceivable options include a message to a central monitoring system, the revocation of critical rights during the session (blocking critical transactions for example), re-authentication of the user or even the immediate termination of the session.

The technical basis for such a system already exists; approaches and algorithms from the field of big data make a valuable contri-

bution here. With a central security architecture based on an entry-gateway approach, the requirements are also met for responding centrally to anomalies that are detected.

The new tasks of the entry gateways

The networking of applications across organizations is usually realized with APIs today. These are web-based interfaces used to exchange data structures in the XML or JSON format. JSON stands for JavaScript Object Notation. As indicated by the name, this is

SERVER-SIDE INTELLIGENCE CAN LEARN THE NORMAL BEHAVIOR OF AN APPLICATION.

a standard developed for the exchange of data structures in the JavaScript environment. The good news is that networked backend applications and JavaScript-based Rich Internet Applications use the same data exchange technologies. That means the gate-

way infrastructures can be used for both application types in principle. This reduces costs on the one hand and simplifies operation on the other.

In case of critical or confidential data, access to the APIs naturally needs to be protected by authentication corresponding to the required security level. As applications become more and more closely integrated and linked, but are not necessarily located in the same data center or on the same server, the individual applications must be able to access the respective other APIs in the name of the end user for the purpose of data exchange. The end user should be asked for consent to this procedure; this is known as user consent. Various processes for this purpose are defined by the authentication protocol OAuth. While the identification or authentication of a user by means of an authorization server is part of these processes as well, it is not standardized itself – in part to enable a choice of authentication mechanisms. After authentication, the authorization server issues the access tokens which ultimately enable the applications to access the API.

THE END USER SHOULD BE ASKED FOR CONSENT, WHEN INDIVIDUAL APPLICATIONS ACCESS OTHER APIS.

The functionality to issue the access token and its verification are ideally installed in a level upstream from the applications. Such an architecture has various advantages:


1. Central: Support for the OAuth protocol does not have to be implemented individually for each application. This saves time and reduces costs as well as making the integration of legacy applications possible in the first place.
2. Decoupled: A secure implementation of the OAuth standard imposes high standards on the administration of the issued tokens. Delivering this functionality in an upstream gateway infrastructure enables the straightforward and secure integration of the existing applications.
3. Modular: The authorization server is designed as a central component used by all integrated applications. This component must be able to integrate all existing user directories while continuing to support the existing authentication methods. A central security infrastructure ensures that the required directories only have to be linked once.
4. Additional functionality: Central access token verification is the ideal basis for central accounting and therefore also for new billing and business models.

The existing web entry gateways and perimeter architectures constitute an ideal basis for utilizing these advantages. We are convinced that today's web entry solutions need to be further developed in the direction of API management. In addition to support for the modern federation protocols (OAuth and OpenID Connect), this also includes additional functionality such as accounting, throttling, efficient filtering of JSON content and the selective restriction of the externally published API functions depending on user permissions.

What does this trend mean for Nevis customers?

We consider incorporating new trends a challenge and an incentive. We want to consistently provide our customers with the technical means required for the optimum support of their business in a dynamic environment. That is why the topics and trends identified here have already been integrated into new product functionality or are being considered in future product development.

Designing the new functions focuses on the gradual migration to new application architectures and the parallel operation of the existing solutions – which means that using Nevis ensures maximum investment security and a solid basis to face future challenges.

(For further information, visit the redesigned Nevis website: <https://www.nevis-security.ch>) 

Stephan Schweizer

Stephan Schweizer joined AdNovum as Nevis Product Manager in 2009 and has been responsible for the Nevis product strategy and distribution ever since. He likes to spend his free time in nature with running shoes on his feet and, as a tennis beginner, working on improving his still somewhat shaky serve.

Thomas Zweifel

Thomas Zweifel, MSc in Computer Science ETH and MAS ETH MTEC, has been with AdNovum since April of 2005. As Senior IT Consultant, he advises customers in matters of IT security and IT strategy as well as managing a team of engineers and consultants. He likes to spend his free time on and under the water.