Secure Applications without Secure Infrastructures

Dieter Gollmann

Hamburg University of Technology, Hamburg, Germany
diego@tu-harburg.de

Abstract. The Internet (together with other communications systems) has become a critical infrastructure in industrialized societies. We will examine to which extent this infrastructure needs to be secured for applications to be deployed securely. We will give examples for application layer attacks that cannot be defended against at the infrastructure layer. Hence, deploying a secure infrastructure is not sufficient to protect critical applications. Conversely, we will give examples where an application can be protected without relying on security services provided by the infrastructure. Hence, deploying a secure infrastructure is not necessary to protect critical applications. We will argue that it is only essential for the computing infrastructure to protect its own execution integrity and for the communications infrastructure to offer availability.

Keywords: Critical infrastructures, application security, security engineering.

1 Introduction

It is today common place to observe that industrialized societies have become reliant on IT to such an extent that the Internet (together with other communications systems) has become a critical infrastructure. It would then seem natural that this infrastructure must be protected against attacks; otherwise we could no longer use the services we have become so accustomed to rely on. This view would be supported by the history of IT security, which has important origins in operating systems security and communications security. Both provide protection at the level of IT infrastructures.

We will argue that such a view is mistaken. Protection of the infrastructure is neither necessary nor sufficient to protect applications deployed on the infrastructure. Society relies in the first instance on the services provided by these applications. Hence, critical applications need to be protected. Protection of the infrastructure is only necessary to the extent required by the application. To be precise, we have to start from a risk analysis for a given application and then decide which attacks are best defended against within the application, and when it is better to rely on security services provided by the infrastructure.

We will illustrate this point with a number of case studies. With the advent of the World Wide Web security functions such as access control started to move

I. Kotenko and V. Skormin (Eds.): MMM-ACNS 2010, LNCS 6258, pp. 21–31, 2010.

from the operating system into the browser. This trend is still continuing. Access control in browsers increasingly resembles access control in a traditional operating system. Attacks such as cross-site request forgery and cross-site scripting cause us to move defences from the browser into individual web pages. We will then briefly cover DNS security and in particular DNS rebinding attacks to discuss which security services should be expected from the infrastructure (DNS, in this case) and where the application should protect itself. Finally, we will discuss attacks on applications that rely on SSL/TLS to show that securing communications may not be sufficient for securing the application. In summary, we will make the case that *security is moving to the application layer*.

2 Browser Security

In the 1970s and 1980s work in computer security had a strong focus on operating system security. The operating system can be viewed as an *infrastructure* component providing users and applications with a file system, managing memory, and managing processes. The security services provided by this infrastructure refer in the main to memory and file management. Processes should not be able to read from or write to memory locations allocated to other processes, unless explicitly intended by inter-process communications. Users sharing a machine should get access to files only if permitted by the policy given (multi-user security). Fundamental security concepts such as status information (supervisor/root and user mode), capabilities, and access control lists were developed in this time.

The attacker was a user with (legitimate) access to the operating system interface trying to enhance his privileges or to get illegitimate access to resources. Security features in an application could typically be disabled by an attacker with supervisor permissions at the operating system level, for example by changing the security settings of the application. In this scenario application security intrinsically relies on the security services supplied by the infrastructure.

2.1 Browser Sandbox

This situation changed in the 1990s when the Internet was opened to general use and the first graphical web browsers emerged. The attacker now was a remote entity using the interfaces provided by network protocols and in particular by the web browser. Access control in the *Java sandbox* could constrain code independently of any security services implemented by the operating system. If we treat the browser as an application running on top of an operating system, we have an instance of an application that includes its own protection mechanisms without relying on security services provided by the infrastructure. The *reference monitor* had moved from the operating system into the browser.

2.2 Software Security

At the same time *software security* deficiencies in the operating system started to attract much attention. A remote attacker could, for example, exploit a *buffer*

overrun vulnerability to run code on a victim's machine [13]. *Ping-of-death* was a denial-of-service attack of the same kind. To defend against such attacks the infrastructure had to be secure in the sense that it could deal with intentionally malformed inputs [9]. In other words, the infrastructure has to guarantee its own *execution integrity* but does not have to supply the application with security services.

3 Web Page Security

A web page is requested by the client's browser through an HTTP request. HTTP cookies included in a request may authenticate the client to the browser. Server-side scripts process request parameters to construct instructions to back-end servers. The response is transmitted from web server to client and rendered by the client's browser. The server may set cookies in a response header. Dynamic web pages contain scripts accepting user input. Scripts may request further server connections. Several attack vectors target this interplay between client and servers.

- An attacker may retrieve cookies from the client, be it to profile the user or to use the cookies to impersonate the client.
- A malicious script in a web page may perform inappropriate operations on the client.
- A malicious script may use the client as a stepping stone to attack a third party.
- A malicious user may send malformed inputs in an HTTP request to perform inappropriate actions with the help of vulnerable server-side scripts (code injection).

3.1 Code Injection Attacks

SQL injection is an example for a code injection attack. A server-side script constructs a SQL query for a back-end database server as a string put together from code fragments that should capture the query logic and from request parameters. Malformed user input in request parameters can change the query logic or insert new database instructions. Note that a single quote terminates strings in SQL. The attacker could thus submit input containing a single quote followed by SQL clauses which would then become part of the query.

To defend against this attack we could either include suitable *sanitization operators* in the script that aim to detect and neutralize malformed inputs. This defence is located firmly within the application. Alternatively, we could modify the infrastructure so that it can protect its own execution integrity. Instead of constructing database queries as strings, queries are precompiled with placeholders for user input. The actual user input is substituted for these placeholders (bound parameters) at runtime.

3.2 Origin Based Access Control

At the client side the browser has become the infrastructure for handling web pages. Today, this infrastructure provides the following security services:

- The browser controls how cookies are included in requests; the widely adopted *same origin policy* states that a cookie may only be included in requests to the domain that had set the cookie.
- The browser controls to which extent a script in a web page may access local memory; in the initial Java sandbox policy a script had no access to local memory; in the Java 2 security model more fine grained access control became possible [8].
- The browser controls where a script in a web page may connect to; again, the *same origin policy* is usually applied to regulate this aspect.

In all three cases the browser performs access control with respect to an origin based security policy. To enforce such a policy, the browser must authenticate the origin of a web page. Current browsers do this in a rudimentary way. They translate between the IP address of the server the page has been received from (more on this in section 4) and the domain name of this server, but there is no fine grained authentication of the individual parts of a web page.

3.3 Cross-Site Scripting and Cross-Site Request Forgery

This shortcoming is exploited by *cross-site scripting* attacks (XSS) [5]. Such an attack uses a 'trusted' server, i.e. a server with more access rights than those granted to the attacker, as a stepping stone. A malicious script might be placed directly in a page on the trusted server (stored XSS, e.g. via a bulletin board). In another version of XSS the script is hidden in a form in a page on the attacker's server. When a victim visits this page a request that contains the hidden script as a query parameter is automatically sent to the trusted server. Should the server mirror this query parameter back to the victim (e.g. in a response to a search) the script is executed in the victim's browser with the access rights of the trusted server (reflected XSS). XSS can be used, for example, to steal cookies from the client.

Authentication of origin has failed as it did not correctly capture the true origin of the attacker's contribution to the page received from the server. Cross-site request forgery attacks targeting a server follow a similar principle [4]. The server has to 'trust' a client, i.e. there has to be an authenticated session (more on this in section 5) where the client has more access rights than those granted to the attacker. The attacker manages to send actions to the server within this session, which are then executed with the access rights of the client.

Client and server could perform authentication at the application layer to defend against this type of attack, rather than relying on the infrastructure provided by browsers and web servers. So-called XSRF prevention tokens are message authentication codes for actions computed from a shared secret that

had been established when the session was created. It is essential to store this secret at the client side in a place out of reach for an attacker able to circumvent the browser's origin based security policies. Once more, the application takes care of security and does not rely on a security service provided by the infrastructure.

4 DNS Security

The Domain Name System (DNS) is, in a nutshell, a distributed directory service managing information about so-called *domain names*. Its core service is the mapping from host names to IP addresses, performed for each domain by one of the *authoritative name servers* for that domain. The DNS is a critical infrastructure for the World Wide Web. Users rely on a correct binding from host names to IP address to get access to the services they wish to use. Browsers rely on correct bindings when enforcing origin based security policies.

4.1 Cache Poisoning

There are two types of attacks that break the correct binding between host names and IP addresses. On one side there are the 'traditional' attacks impersonating an authoritative name server to forge IP addresses in the domain of that server. *Cache poisoning attacks* exploit certain features of the DNS, including the caching strategy of resolving name servers and a challenge-response authentication that relies only on the unpredictability of challenges, to achieve this goal. A particularly effective cache poisoning attack using so-called additional resource records is due to Dan Kaminsky[1]. Defences against cache poisoning attacks can be provided at the infrastructure level, e.g. by running separate resolving and authoritative name servers in a domain, by designating random ports for replies from the authoritative name server as to increase unpredictability, and ultimately by having the response from the authoritative name server digitally signed (DNSSec, RFC 4033 to RFC 4035 [1,2,3]).

4.2 DNS Rebinding

There is a second type of attack where an authoritative name server is the source of incorrect bindings. Such *DNS rebinding attacks* and were first discussed in [6]. DNS rebinding attacks exploiting features of browser plug-ins are described in [10]. With DNS rebinding the attacker circumvents origin based policies in the client browser. For example, a script from a page hosted by the attacker may connect to a victim's IP address the browser accepts to be in the attacker's domain because it has been told so by the attackers authoritative name server.

The client browser would have to double check with the host at the designated IP address whether it considers itself to be in the attacker's domain. It must also

[1] For details see e.g.,
`http://unixwiz.net/techtips/iguide-kaminsky-dns- vuln.html`

be noted that it is an intrinsic problem if the client accepts policy information from a third party without checking its veracity.

More generally, we may ask whether it is necessary for an application to rely on the DNS to provide an authenticated binding between a name (not necessarily a domain name) and an IP address. Alternatively, we could split the task of a *rendezvous service* that binds a name to an unauthenticated IP address from the task of an *authentication service* verifying that an address given indeed belongs to that name. The security property expected from the infrastructure would then be availability, which might be achieved by running multiple independent rendezvous services. Address authentication could be implemented in the application layer, e.g. based on secrets shared between client and server such as a user password.

5 Secure Sessions

Besides operating systems security, communications security has been the second main pillar of information security. Protocol suites such as SSL/TLS (TLS v1.2, [7]) or IPsec [11] facilitate the establishment of secure channels between two parties that are connected via an insecure network. More precisely, the threat model assumes an attacker that can read, delete, insert, modify, and replay traffic; direct attacks against end systems are, however, not considered.

In the 1990s distributed applications were 'secured' by running the application over SSL. *https* is a prime example for this pattern: a secure web page is a page accessed via an SSL/TLS channel. Application security builds directly on security services provided by the communications infrastructure.

This approach has two shortcomings. Many end systems are not well secured. This invalidates one major assumption of the threat model that underpins traditional communications security. Arguably, it is more realistic to assume that the communications system is secure but current end systems are not, rather than the other way round. Section 2 has already hinted at this problem. Secondly, attempts at linking concurrent sessions established at different protocol layers may fail.

Consider the following procedure for establishing a mutually authenticated application layer session between a user and a server that share a secret password. First, the user's client establishes an SSL/TLS channel with the server (host). In this step the client's browser checks that the distinguished name in the server certificate matches the host visited and that the certificate is still valid. The user then sends the password via the SSL/TLS channel; the server authenticates the user and returns a HTTP cookie to the client. This cookie is included in future requests issued within the application layer session. The server takes the cookie as evidence that the requests are coming from the user previously authenticated. The EAP-TTLS protocol gives a concrete implementation of this authentication pattern (Figure 1).

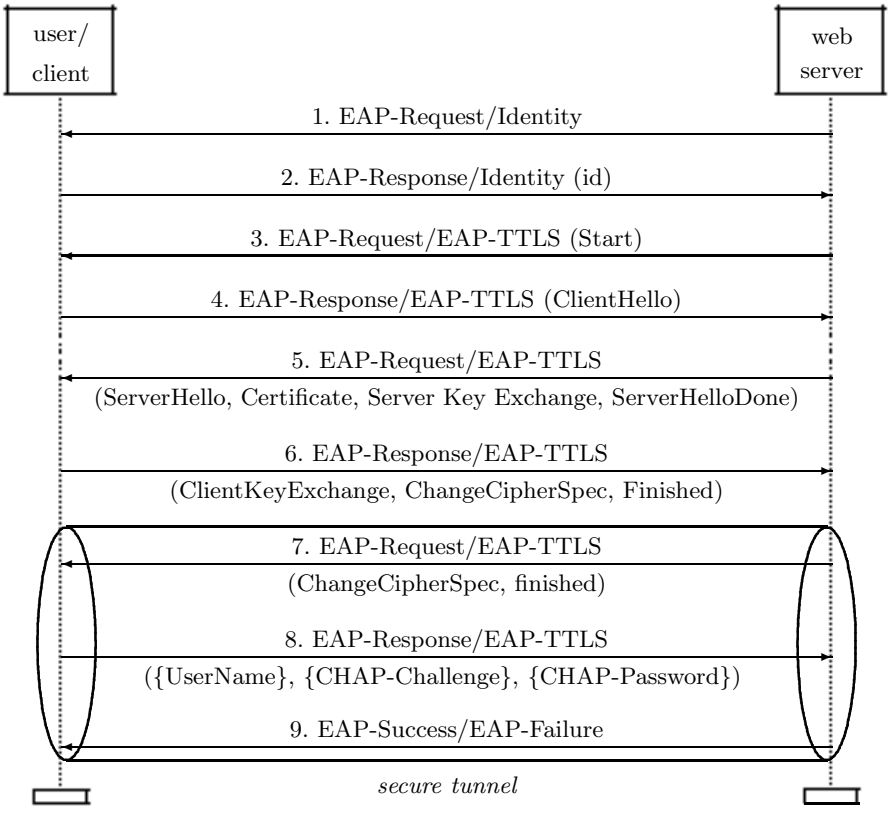

Fig. 1. EAP Tunneled TLS: EAP-TTLSv0 with CHAP

5.1 Man-in-the-Middle Attacks

A protocol such as EAP-TTLS achieves its goal as long as the SSL/TLS channel has as its endpoint the server holding the password. This is not guaranteed by the protocol itself. Server authentication during the SSL/TLS handshake just guarantees that the server has a valid certificate. It is up to the user to make sure that the host is the one intended.

This check is not always straightforward; host names are not always indicative of service offered. Furthermore, there exist various ways of luring users into connecting to the wrong server. For example, an attack[2] targeting traders with the German Emissions Trading Authority (DEHSt) started from an email purporting to come from a security manager requesting an upgrade to improved security standards[3].

[2] First report on `http://www.ftd.de/unternehmen/finanzdienstleister/:gestohlene-co2-zertifikate-hacker-greifen-emissionshaendler-an/50069112.html`

[3] This mail – in German – can be found at `http://verlorenegeneration.de/2010/02/03/dokumenation-die-phishing-email-im-emissionshandels-hack/`

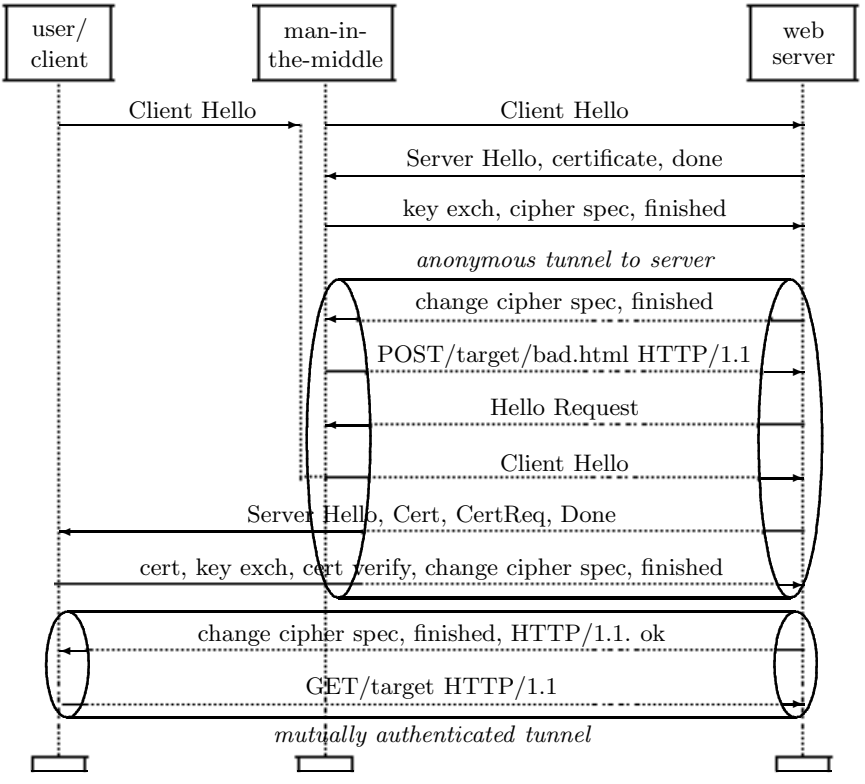

Fig. 2. Man-in-the-middle attack exploiting TLS session renegotiation

Once a user is lured into establishing an SSL/TLS channel with the attacker, the attacker can act as a man-in-the-middle establishing its own SSL/TLS channel with the server. Authentication requests from the server are passed on to the user; the user's response is forwarded to the server; the cookie from the server is sent back to the man-in-the-middle who now can hijack the user's application layer session. A possible countermeasure are cookies tied also to the SSL/TLS channel as proposed in [14]. In the presence of a man-in-the-middle attack client and server use different SSL/TLS channels and could thus detect that cookies are not received in the same channel as they had been originally sent.

A man-in-the-middle attack in time is described in [12]. It exploits a particular usage of SSL/TLS for controlling access to protected resources on a web server. Here, client and server are in possession of certificates. The client initially gets anonymous access to a secure web site by establishing an SSL/TLS channel with server authentication only. When the server receives a request for a protected resource, SSL/TLS session renegotiation is triggered with a *Hello Request* message. In the new session the server asks for client authentication.

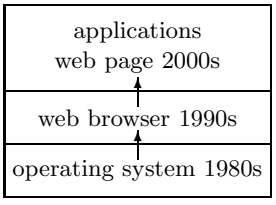

Fig. 3. The reference monitor is moving into the web page

In the man-in-the-middle attack (Figure 2) the attacker waits for a session initiation from the client. The client's message is suppressed and the attacker starts its own session with the server. The attacker sends a request for a protected resource (in Figure 2 a web page is posted to the server) whereupon the server triggers session renegotiation. From this time on the attacker acts as relay between client – that is in the process of establishing a new channel – and server until both have established a new mutually authenticated SSL/TLS channel. A request sent in this new channel will be attributed correctly to the authenticated user and executed with that user's access rights. The attacker's HTTP request had been constructed so that it would be a prefix to the next request in the current session and will now also be executed with that user's access rights.

Note that RFC 5246 does not promise any link between sessions when defining TLS renegotiation. Application designers who had used renegotiation to 'upgrade' the authentication status of the client had thus assumed a service not provided by the infrastructure. To address this situation, RFC 5746 [15] defines a TLS extension where renegotiations are cryptographically tied to the TLS connections they are being performed over. In this case, the infrastructure has followed to meet the – initially unwarranted – expectations of an application.

6 Conclusion

Security is moving to the application layer. Once, the design of secure operating systems and Internet security protocols were the main foundations of information security. According to the then predominant mood, IT systems could be used securely once secure infrastructures were in place. Remnants of this era can still be found in claims that one MUST secure operating systems or the Internet to be able to securely use today's critical IT infrastructures.

We observe, however, that security mechanisms in end systems are moving to the application layer of the software stack. The reference monitor has moved from the operating into web pages (Figure 3). Security components at upper layers may be effective without support from below. This works as long as direct access to the lower layers need not be considered as a threat. In parallel, communications security mechanisms have been moving to the application layer of the protocol stack. At the point where end system security and communications security meet, i.e. in the software components running network protocols, we

have seen shared secrets moved up to the application layer to defend against attacks at the infrastructure layer (Section 3.3), contrary to the conventional security strategy that tries to embed secrets at a layer as low as possible, e.g. in tamper resistant hardware.

The security services expected from the infrastructure may thus change over time. We can also observe that our view of what constitutes the infrastructure may change over time. The web browser that started as a new application has today become an essential infrastructure component for Web services.

The Internet is a critical infrastructure because it is the platform for critical applications. Our primary challenge is the protection of these applications. Security services provided by the infrastructure may help in this cause, but trust in these services may also be misplaced when application writers misunderstand the security properties actually guaranteed.

It is a trivial observation on security engineering that defenders ought to know where their systems will be attacked. When attacks are launched via the interface of web applications, the first line of defence should be at that layer.

Attacks may be directed against the application, e.g. fraudulent bank transfers in an e-banking application. Application-level access control necessarily relates to principals meaningful for the application. There may be mappings from those principals to principals known to the infrastructure so that security services from the infrastructure can support application security. However, in every instance we must verify that it is not possible for attackers to redefine the binding between principal names at different system layers. We may thus surmise that it is more likely to find access control solutions at the application layer, as borne out by our earlier observations on reference monitors. In this respect, we have secure applications without a security infrastructure.

Attacks may be directed against the end system hosting the application. Software vulnerabilities in an application may present the attacker with an opportunity to step down into the infrastructure. Although software security issues could be addressed in each application, it would be desirable to have a 'secure' computing infrastructure, i.e. an infrastructure that can deal with malformed inputs forwarded via the applications. In this respect, critical applications benefit from a computing infrastructure that can protect its own execution integrity.

The primary property required from the communications infrastructure is availability. Security services such as confidentiality, integrity, or authenticity may or may not be provided by the communications infrastructure. The relative merits of delivering these services in the various layers of the network stack have been discussed extensively in the research literature. The most relevant issue for critical applications are the choice of relevant principals that can serve as logical endpoints for application layer transactions, and the authentication of those principals.

We leave the reader with a final challenge. When security is moving to the application layer, responsibility for security will increasingly rest with application writers and with end users. At this point in time, neither of the two communities is well prepared to take on this task, but nor has security research made much progress in explaining to non-experts the implications of security decisions.

References

1. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: DNS security introduction and requirements. RFC 4033 (March 2005)
2. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: Protocol modifications for the DNS security extensions. RFC 4035 (March 2005)
3. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: Resource records for the DNS security extensions. RFC 4034 (March 2005)
4. Burns, J.: Cross site reference forgery. Technical report, Information Security Partners, LLC, Version 1.1 (2005)
5. CERT Coordination Center. Malicious HTML tags embedded in client web requests (2000), `http://www.cert.org/advisories/CA-2000-02.html`
6. Dean, D., Felten, E.W., Wallach, D.S.: Java security: from HotJava to Netscape and beyond. In: Proceedings of the 1996 IEEE Symposium on Security and Privacy, pp. 190–200 (1996)
7. Dierks, T., Rescorla, E.: The TLS protocol – version 1.2, RFC 5246 (August 2008)
8. Gong, L., Dageforde, M., Ellison, G.W.: Inside Java 2 Platform Security, 2nd edn. Addison-Wesley, Reading (2003)
9. Howard, M., LeBlanc, D.: Writing Secure Code, 2nd edn. Microsoft Press, Redmond (2002)
10. Jackson, C., Barth, A., Bortz, A., Shao, W., Boneh, D.: Protecting browsers from DNS rebinding attacks. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 421–431 (2007)
11. Kent, S., Seo, K.: Security architecture for the Internet protocol, RFC 4301 (December 2005)
12. Marsh, R., Dispensa, S.: Renegotiating TLS. Technical report, PhoneFactor Inc., Malvern (November 2009)
13. One, A.: Smashing the stack for fun and profit. Phrack Magazine 49 (1996)
14. Oppliger, R., Hauser, R., Basin, D.A.: SSL/TLS session-aware user authentication. IEEE Computer 41(3), 59–65 (2008)
15. Rescorla, E., Ray, M., Dispensa, S., Oskov, N.: Transport layer security (TLS) renegotiation indication extension, RFC 5746 (February 2010)