Threats to internal application security

Yousef Ibrahim DARADKEH
College of Engineering at Wadi Addawasser, Prince Sattam bin Abdulaziz University
daradkehy@yahoo.ca

Svetlana Mikhailovna ARISTOVA
Kudymkar affiliation of the Udmurt State University
s.aristova@mail.ru

Petr Mikhailovich KOROLEV
Studia Korolevae Int
studiakorolevae@mail.ru

***Abstract* — Practice on development the business applications for companies are presented; issues of audit threats to information systems counterpointed as good and bad news for business and engineers. Ways to exploit the vulnerability discussed. Audit is to minimize potential damage to the company from incorrect performance of the software that supports a given business process. What should we worry about with internal applications? Authors of the paper do answer on this.**

***Index Terms* —** Auditing custom-made application security, threats to business process, application threat model**.**

INTRODUCTION

Various companies' information systems no longer simply mirror offline business processes. Many of transactions that describe corporate business systems' users' actions have no physical analogues at all. Plenty of documents have no hard copy backups. All of this transforms software from a helpful tool to an integral component of the business. For some businesses, these components are more important than the physical ones: a telecom, bank or store simply cannot function without their billing, banking and registry systems respectively. On the other hand, an oil company can easily continue pumping oil. They just won't be able to keep track of it very well.

Therefore it should not surprise you that threats to business process functionality are dealt with very seriously. Different companies approach their applications differently: there are the most critical ones, with the company's functions as a business unit depending on their performance. For example, at one oil company, we have discovered a bizarre incident classification metric. It flagged halting the application responsible for sending the oil shipping signal with the same flag as a terrorist act at the pipeline or an assassination attempt at an executive's life.

The damage from a sudden halt of an application heavily depends on how competitive the business environment it's found in is. For example, halting an online store will lead to customers simply buying their products at a different one, leading to direct losses. Conversely, incorrect operation of internet banking will not lead to direct losses, since all of the bank's clients can't just up and leave. They'll just wait until the online portion is up and running again. Some of the bank's operations might be performed offline at the bank's office, but the bank would not lose money. However, indirect losses would still be there: there would be an increase of stress on their call centers and tech support; emotional clients will complain about the bank on social networks; for a small percent of these clients, this would be the last straw, and they would switch banks; and so on. If we're talking about monopoly services, such as railroads or civil services, a long (Korolev et al 2017; Daradkeh et al 2016)

We think about the approach to vulnerability, in such a turn to find the problem points. The vulnerability is that it has some place of entry, into which you can enter something that stings. It can be any open system. The closed system is invulnerable. Communication opens systems, that is, communication is a condition of vulnerability. Any hole through which the exchange between the internal and external can occur is a point of vulnerability. Any publicity is a condition of vulnerability. If there are substances that are outward or inward, which cause vulnerability, they are able to spread and sting other systems in the open space, or, spreading inside, to injure the system in which they exist. Invulnerability appears when the system has immunity, that is, a means of neutralizing the vulnerability substances. Or when it is possible to disconnect the uninvited system from the external environment. Infection can be the goal of a particular activity that seeks to injure the system in order to either paralyze or destroy it. The susceptibility and immunity to an external signal must be in a certain ratio. Sometimes use means of recognizing the threat of wounding. But these funds themselves as standards can contain vulnerabilities. Thus, the communicating systems in principle cannot be pure, free from the substances of wounding. And they require for their hygiene the availability of filters. The article deals with issues that can be treated as good news and as bad news in protecting information systems from vulnerability threats. When merging and absorbing, it is also important to put systems in the same state, otherwise the "predator" will swallow the "victim", and instead of combining the systems, a system of "saturated predator" will be obtained. Six samples of vulnerabilities, such as buffer overflow in path processing, integer overflow in path processing, determining of a path exists, no auditing on requests, deny access control entries handled incorrectly authors of the book collect in table fields that include ID, Name, Description, STRIDE classification, DREAD rating, Corresponding threat, and Bug. [1:240-242]

PERFORM AN AUDIT AND BUILD AN APPLICATION THREAT MODEL.

To minimize potential damage to the company from incorrect performance of the software that supports a given business process, we must perform an audit. We have to test the software for security, function, stress and penetrability. There are companies that focus on providing these services, while some companies large enough have dedicated labs that test an application's performance before launching it. In order to optimize the process of testing and analyzing the protection of the software, we have to build an application threat model.

Most corporate and ministry applications are built from scratch, or tailored to the company from a specialized platform, so we will refer to them as "custom-made" (as opposed to "end-point"). They are not simply finished after a single development cycle. Instead, they are constantly updated and tweaked based on business or regulatory changes, and external or internal economic or political demands. Therefore, a custom-made application cannot be protected in a single stroke: it must be updated to keep up with the changes forced on the software by new tax codes, or an entry of a foreign competitor on the market, or the release of a new software package. With every new change, the application must have a security audit [3].

THE GOOD NEWS

3.1 Use the application in a way that the developers did not plan for

Application security auditing began with the rapid expansion of the Internet. Before the Internet, a business application (why yes, there were business applications before the Internet) would have no more than a hundred named users. As soon as web applications became popular, almost any Internet user could gain access to them. Suddenly, an application connected to the Internet could have an enormous amount of anonymous users. Part of these users, inspired by their anonymity, could easily try to use the application in a way that the developers did not plan for.

Applications connected to the Internet first gave rise to attacks that targeted them, and, as a reaction – to various security methods and systems that protected them. As soon as web applications became popular, security auditing started to focus on all application levels: interfaces, settings, source code, etc. It stopped being a hobby, and started being automated with dedicated tools[1].

We will not digress too much into the fine points of the classifications and industry standards. Suffice to say that most of these vulnerabilities are completely irrelevant for internal applications.

3.1.1 Incorrect processing of user input data.

If the application is connected to the Internet and lacks user input sanitization, it lets the user perform a variety of injection attacks. Basically, the user can input some code instead of the input the program needs (such as a username), and send a request to the database that can grant him information or even write access. This vulnerability is a massive risk that can lead to data loss or denial-of-service attacks: surely there are a handful of malicious users among the millions that have access to the page that are willing to try it.

3.2 To exploit the vulnerability, one needs three things

For an internal application (such as an electronic document flow system, logging system, call center support, ERP, CRM, billing and so on), however, this vulnerability is a non-issue. First of all, there are very few users with access, and all of them have gone through authorization before gaining it: they entered the building with their pass, logged in with their login and password or with their e-key, and so on. Therefore, despite the fact that the user can technically do the attack, the chances of it happening are very slim. In order to exploit the vulnerability, you need three things: you have to find it; you have to have the technical knowledge to exploit it; and most importantly, be certain you could get away with it. Those three factors are present for most external users to at least some amount, and eventually a user with all three will be found. For internal users, however, combining all three is a massive coincidence. Even if the user has found the vulnerability (either manually or with a scanner) without his system administrator or coworkers noticing it, he might not be able to exploit it. Most business system users are clerks: salespeople, accountants and managers. Very few of them have hacking skills. But even if the user does have access *and* the skills[2], they would probably not risk running the attack, since they know they would be caught and punished.

3.3 No access to the whole picture.

Therefore the classical approach to web application security is redundant for internal applications, and does not show the whole picture. If you look at the whole spectrum of business applications in a large company, you would see that most of its applications are internal, making their users employees and authorized external users, even if they use a web-interface. This does not only include accounting, ERP, CRM and document automation, but also the banking user interface when doing online banking, or the checkout screen at an online store: the user is authorized and a single person is responsible for their action. The company has the person's data, which severely limits the risk of exploiting the vulnerability.

4 THE BAD NEWS

And now we come to the bad news: we can't use the common web application security scanners for internal applications, even if they use a web interface. The results would be of no help, since they wouldn't show any of the actual threats. But this is not the only piece of bad news.

[1] Despite the powerful force that web applications have played in application security auditing, it did lead to some drawbacks by entrenching threat modeling for business applications in general. For example, threat modeling for any application now begins with the OWASP methodology (Open Web Application Security Project, www.owasp.org), which was initially created for web applications. OWASP Top 10, an annual vulnerability classification list has become so entrenched in application security that it is now a requirement for many regulators, such as the PCI Council (the consortium responsible for PCI DSS and PA DSS standards).

[2] which is rare: a hacker can make a lot more money than an accountant, so it's unlikely that on would work as an accountant

What should we worry about with internal applications?

4.1 First and foremost, a break in stability.

An incorrect request can do more damage than a hundred hackers by simply hanging the application for a long time. As I've mentioned earlier, the accessibility of a business application is its primary characteristic – far more important than integrity or confidentiality. A break in integrity or confidentiality does not always lead to a direct loss, while inaccessibility of data or a hung application always does.

4.2 Elevating access privileges and breaking access control

Second of all, we should worry about elevating privileges in an application that could give the user access to information that the user role should not be able to view. I use the word "view" intentionally, since an internal user sometimes only needs to see a few phrases or numbers on the screen to start problems. It could be the salary of a colleague or his boss, which could trigger massive jealousy; information about VIP clients that could be of use to journalists; intellectual property and trade secrets that competitors would gladly pay for; revenue and losses of a publicly traded company before they are revealed of interest to stock brokers; and so on. You know how confidential some information in your company is, to the point that it's hidden from your employees. Elevating access privileges and breaking access control can be done in hundreds of ways: from debugging triggers left over from development, to specifically introduced developer backdoors.

Since we're already stroking your paranoia, we should also mention vulnerabilities that could lead to forgery and scams, in other words, those that break data integrity.

For example, there are urban legends about a few lines of code in banking applications that keep track of all the rounding errors, and send them to a specific account that the developer controls. We have not personally seen any, but it would definitely be possible to write something like that.

4.3 Uniqueness of threat

Most of those threats to internal application security are unique to each business application and process. We can't find them using out-of-the-box or cloud-based scanners. Even the most basic backdoor, a hidden administrator password hardcoded into the source code, depends on how a given application calls the authorization function.

4.4 Network traffic analysis tools wouldn't help you

Third piece of bad news. Exploiting these vulnerabilities for internal applications is processed as completely legitimate transactions by the systems themselves. Therefore, network traffic analysis tools wouldn't help you. If most of OWASP Top 10 vulnerability exploitations can be tracked by network or semantic anomalies (basically, you would see a stream of data that is odd for a given operation), internal vulnerabilities show up as benign to those scanners.

CONCLUSION

All of this adds up to making the process of finding these vulnerabilities and getting rid of them is a joint process between the client who must know the application inside and out, and code auditors who understand the vulnerabilities, and know how to get rid of them.

Auditing custom-made application security is a relatively young industry, but based on how IT is currently developing, it should develop very quickly. New programming languages (Apple has recently launched SWIFT, Google has released Go, etc.), new automated objects (wearable electronics, cars, household and medical items, etc.) with their own interfaces stimulate a push to custom-made development, adding new developments to business processes, and requiring more security auditing.

It is very important that you do not rely on established threat models, but use them only for threats that are a concern to your application.

ACKNOWLEDGMENTS

We thank for permission to use excerption of the text from the book [2] published at Lambert Academic Publishing

REFERENCES

[1] F.Swiderski, and W.Snyder, Threat modeling. Microsoft Press, 2004.

[2] P.Korolev, Y.Daradkeh, and S.Aristova, Connecting Known and Unknown. LAP Lambert Academic Publishing, 2017 Available: https://archive.org/stream/brosh17-3/brosh17-3_djvu.txt

[3] Y.Daradkeh, S.Aristova, and P.Korolev. "Observation and Audit of the Processes in Experiences with Uncertainty", *J Comput Eng Inf Technol* DOI: 10.4172/2324-9307.1000160. Available: https://www.scitechnol.com/peer-review/observation-and-audit-of-the-processes-in-experiences-with-uncertainty-VFQT.php?article_id=5661