Университет по библиотекознание и информационни технологии

Курсова работа
по
Защита на информацията
на тема

Защита на информацията в уеб пространството

Изготвена от Иван Казмин,
студент в УниБИТ, 4. курс, Фак. №2043
ivan.kazmin@gmail.com

Съдържание

Използвани термини и съкращения

MAC (message authentication code) – информация с малък обем, която се използва за удостоверяване на съобщение и за осигуряване на цялостност (непроменост) и автентичност (произход)

ключ (в контекст на криптография) - секретна информация, използвана от криптографски алгоритъм при шифриране/дешифриране на съобщения

хеширане - „раздробяване“, сгъстяване на дадено множество чрез използване на еднопосочна, математическа функция. Оригиналното съобщение не може да бъде получено от изходното съобщение

фишинг - форма на онлайн атака от фалшиви уебсайтове, създадени да дублират сайта на вашата банка или други подобни, които посещавате често. Целта им е да ви подведат да въведете чувствителна за вас информация - като пароли, номер на кредитна карта и други

Увод

В днешно време почти всяко наше електронно устройство е свързано с Интернет, било то преносим компютър, мобилен телефон, таблет и т.н. Наивно е да се счита, че с появата на Интернет няма да се появят и **редица проблеми, породени от злонамерени действия, насочени от потребител към Интернет и обратно**, общо казано. Това налага да се вземат мерки ***да се защитят виртуалното пространство и неговите обитатели***, които често небрежно, разсеяно и невнимателно прекарват време там. **Този материал има за цел да запознае своите читатели с част от проблемите по сигурността в уеб пространството**. Поради мащабността на темата, курсовата работа няма за цел да засегне тези проблеми изчерпателно, а да въведе читателя в тях и да опише техните решения.

Защита на информацията по направление клиент-сървър и обратно

Посещавайки даден уеб сайт, ние постоянно обменяме информация със сървъра, на който той се намира. Често уеб сайтът изисква от нас да въведем дадена информация - **лични данни, потребителски имена, пароли, мултимедийни данни** (снимки, видео клипове), **номер на кредитна или дебитна карта**, както и често уеб сайтът изпраща информация на нас. **Очевидно съществува проблем с тази практика** - почти винаги информацията, която си обменяме с даден сървър, е **чувствителна, лична, конфиденциална** и затова тя би била обект на интерес на трети лица. Комуникирайки си с един сървър на мрежово ниво, информацията преминава и през други сървъри, докато стигне до определения получател. **Интернет е мрежа от сървъри, при която не всички имат пряка връзка един с друг.** Ако пътуваме от София до Берлин с автомобил, трябва да преминем през Ниш, Белград, Будапеща, Братислава и Прага. Аналогично, комуникацията ни с един сървър преминава през други сървъри-посредници, които могат да я **прочетат в нейния суров вид и дори да я променят, което я прави уязвима**. За щастие е намерено решение на този проблем и то се нарича **TLS - Transport Layer Security, наследникът на SSL - Secure Sockets Layer**.

Преди да навлезем в това какво е TLS и как работи, ще се запознаем както с някои термини, така и с това **какво е криптирането** и два негови класа - симетрично и асиметрично криптиране.

Криптография и шифроване

Криптографията е наука, чиято задача е да гарантира обезпечаване сигурността на различни като съдържание и обем информационни масиви. Основната задача на криптографията е намиране на решения в четири основни области, както следва:

- *Конфиденциалност на информацията*
- *Идентификация на участниците в процеса на обмен на информация*
- *Съхраняване целостта на информационните масиви*
- *Контрол върху лицата, участващи непосредствено в процеса на обмен на информация*

Един от основните процеси е този на **шифроване на информацията**. Шифроването е процес, при който данните се преобразуват по начин, който не позволява те да бъдат разчитани, независимо дали става дума за текстова, аудио, видео или друг тип информация. В процеса на шифроване могат да се използват **ключове**, чието предназначение е да кодират и декодират информацията, гарантиращи по този начин нейната конфиденциалност.

Системите за криптиране (криптосистеми), работят по определена **методика (процедура)**, Тя се състои от един или повече **алгоритми за шифроване (математически формули)**; **ключове**, използвани от същите алгоритми, **системи за управление на ключовете**, **незашифрован текст и зашифрован** (кодиран текст или шифрограма).

Симетрична криптография

При нея се използва един ключ, с помощта на който се извършва както шифроването, така и разшифроването на информацията при прилагане на един и същи алгоритъм за двата процеса. Ключът се предава на

двамата участници в процеса при установени правила за сигурност преди обмена на информация. Той трябва да е в тайна и ако се изгуби, информацията не може да се декриптира. В практиката се означава като **методология със секретен ключ**.

Асиметрична криптография

Асиметричната криптография, за разлика от симетричната, използва **два ключа - личен ключ**, който остава в тайна и се знае само от един участник, и **публичен ключ**, който се споделя с всички участници. Тези ключове се наричат още двойка, между тях има математическа връзка и те трябва да се използват заедно за успешно шифроване и дешифроване. В практиката асиметричната криптография е известна като методология с **открит ключ**.

Transport Layer Security

TLS е един от най-използваните протоколи в Интернет. Базира се на SSL, който е предишният протокол, разработен от Netscape за техния Navigator браузър. **TLS обикновено се използва за защита на протоколи като HTTP, FTP, SMTP** и др.

TLS протоколът позволява на клиент-сървър приложения да комуникират сигурно по незащитена мрежа, каквато е Интернет по начин, който предотвратява опити за подслушване и промяна на потока на информация. *Използва се асиметрична криптография за защитена обмяна на ключове между клиент и сървър и след това използва симетрична криптография за криптиране на информацията*, която се обменя. **Message Authentication Codes** (MACs) се използват за **цялостност на съобщенията** (integrity).

При HTTP протокола е нужно клиентът да покаже по някакъв начин на сървъра дали желае да установи защитена TLS връзка или не. Съществуват обикновено два начина за постигане на това:

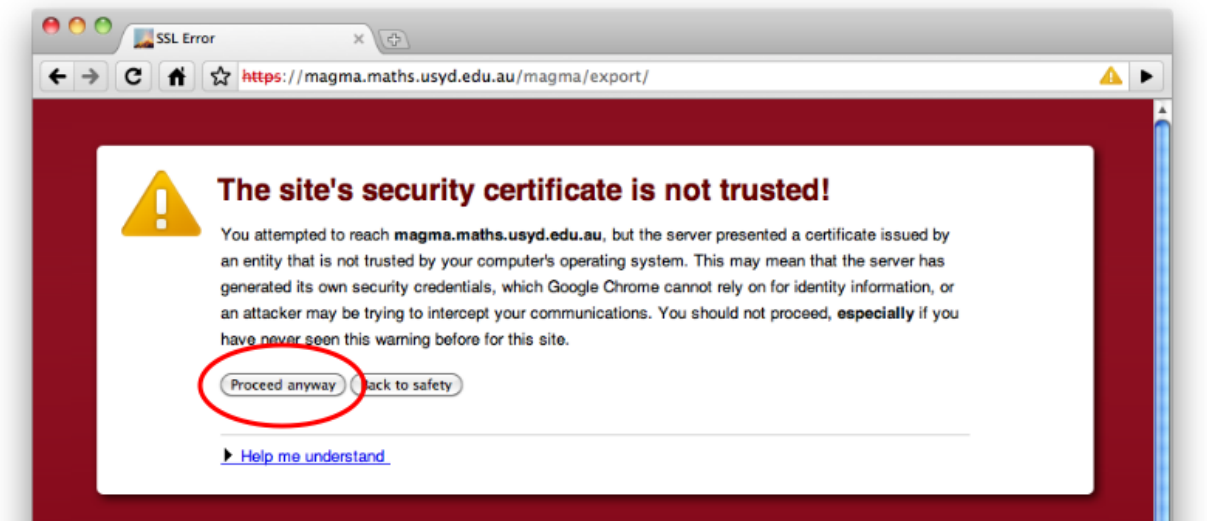
1. *Защитената и незащитената връзка работят на два различни порта. Пример за това е HTTP протоколът, който стандартно използва 80-ти порт. Когато TLS е включен, използва 443-ти.*

2. *Използване на един и същи мрежови порт и за защитена, и за незащитена комуникация. Клиентите могат да заявят на сървъра да превключи на TLS защитена връзка по време на ръкостискането (handshake, обяснено по-долу). Разширението на протокола, който позволява това, се нарича STARTTLS.*

След като клиентът е поискал използването на TLS за комуникация, започва **процедура по съгласуване на различни условия и параметри, чрез които се осъществява сигурносттта на връзката, наречена ръкостискане (handshake).** Пример за TLS ръкостискане се състои от следната поредица от действия:

1. TLS ръкостискането стартира с фаза на преговори. Клиентът изпраща ***ClientHello*** съобщение до сървъра. Това съобщение съдържа определени параметри, като например най-новата TLS версия, която клиентът поддържа, както и списък с поддържани функции за шифроване и хеширане.

Тогава сървърът изпраща ***HelloServer*** съобщение, което съдържа избраните функции за шифроване и хеширане и версията на TLS, както и произволно генериран номер. Сървърът изпраща също и своя сертификат. **Сертификатът** обикновено съдържа информация за органите, които са го



Пример за недоверен сертификат - Google Chrome браузърът не успява да провери организацията, която издава сертификата, и предупреждава за несигурност в защитата. Дава се възможност на потребителя да продължи, въпреки това.

издали (*Certificate Authority*), името на сървъра и публичния ключ, с който ще се криптира връзката. **Браузърът на клиента автоматично проверя-**

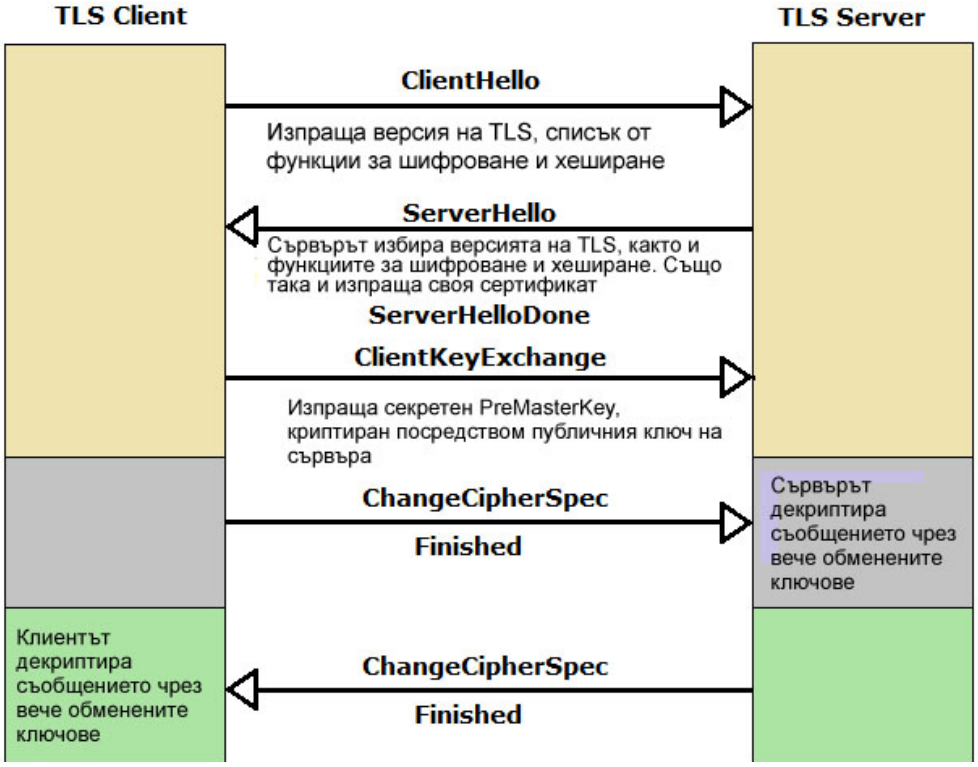
ва валидността на сертификата и уведомява потребителя, ако той е несигурен. След това сървърът изпраща съобщение ***ServerHelloDone***, което слага край на първоначалното ръкостискане.

Клиентът изпраща **PreMasterSecret** номер, който е криптиран чрез публичния ключ на сертификата на сървъра. Това се изпраща заедно с **ClientKeyExchange** съобщение.

Най-накрая, клиентът и сървърът използват вече обменените произволен номер и **PreMasterSecret** номер, за да генерират данни, наречени **Master Secret** (секрет). В този момент фазата на преговори приключва.

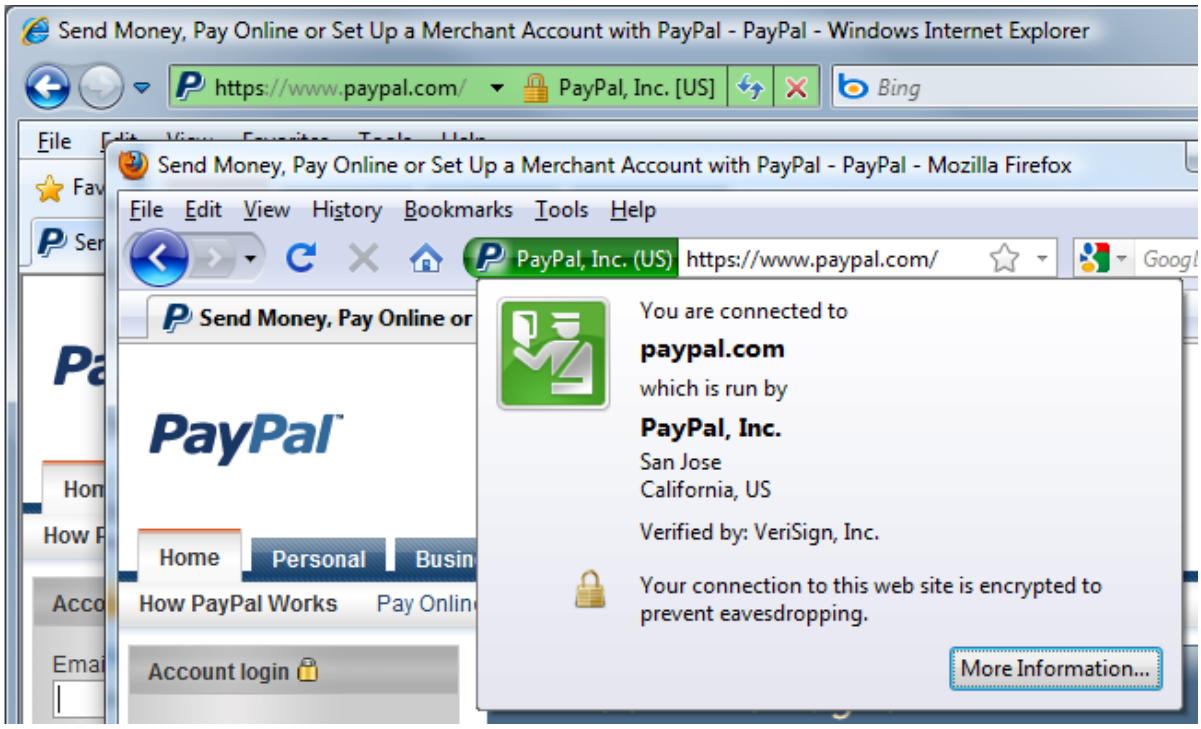
2. Клиентът изпраща ChangeCipherSpec съобщение, което казва на сървъра *от сега нататък информацията, която аз ти изпращам, ще бъде криптирана*. Най-накрая клиентът изпраща Finished съобщение, което е криптирано. Ако сървърът не успее да декриптира и провери това съобщение с вече обменените секрети, връзката се прекратява.

3. Сървърът изпраща ChangeCipherSpec съобщение, с което казва на клиента *от сега нататък информацията, която аз ти изпращам, ще бъде криптирана*. Сървърът изпраща своето криптирано ***Finished*** съобщение, което клиентът декриптира и проверява и това слага край на ръкостискането.



Ръкостискането създава сесийни ключове. Те се използват, за да криптират и декриптират данни, изпратени между клиента и сървъра, като се използва симетрично криптиране. Функциите за шифроване се използват за криптирането на данните. След приключване на предаването на данните, защитеният канал се прекъсва и използваните ключове се изтриват.

Уебсайтовете, които работят с този протокол, обикновено имат URL с префикс „https:“, вместо „http:“. Много е важно SSL сертификатът да е цифрово подписан от доверен **Certificate Authority**. Всеки може да създаде сертификат, но за да се приеме от браузъра, той трябва да е от доверена организация (*Certificate Authority*). Браузърите съдържат в себе си списък с такива организации. За да се приеме такава организация за сигурна и одобрена, то тя трябва да отговаря на редица стандарти. Когато посетим сайт, който използва TLS/SSL, виждаме жълт катинар и/или зелена адресна лента.



Сайтът за Интернет разплащания PayPal е защитен от сертификат на VeriSign Inc. (Trusted Security Authority), заради което префиксът на адреса започва с https:// и част от адресната лента е в зелено. Показана е и информацията за сертификата.

Недостатъци на TLS/SSL

- *Инсталиране на сертификат не е безплатно, тъй като той се закупува от Certification Authority, който трябва да е доверен (trusted).*
- *Криптирането използва повече ресурси както на сървъра, така и при клиента. Това прави комуникацията по-бавна. Увеличава се и трафикът и времето, нужно за отговор от сървъра - може да се увеличи с от два до десет пъти. Затова е добра практика страници, които нямат нужда от защита, да не са защитени.*

Заключение

Недостатъците на TLS/SSL се пренебрежими на фона на предимствата. От гледна точна на уеб потребителя, доверието към един уеб сайт нараства, ако той знае, че уеб сайтът е защитен с протокол за криптиране, особено ако е нужно изпращане на лична информация. Криптографските протоколи са изборът за превантивна и ефективна защита на уеб бизнеса.

Защита на уеб сайт от атака на потребител

Като собственици на уеб сайтове, ние **трябва да подсигурим не само потока на информация към клиента и обратно, но и данните, които се съхраняват в нашите информационни масиви**. Класифицирана информация, която се съхранява на даден уеб сървър с цел използването и от потребители с достъп, би била обект на злонамерени действия - прочитане на информацията, променяне, изтриване. Самият уеб сайт би бил мишена на потребители, които желаят да преустановят и затруднят работата му - било то за лично удоволствие или ред други, по-бизнес ориентирани причини.

Атаките срещу уеб приложенията може да струват много време и пари на организациите, както и да доведат до скъпи и неприятни пробиви на сигурността на данните, което прави изчерпателните стратегии и механизми за защита задължителни. Ще разгледаме някои често срещани методи на компрометиране на уеб система.

SQL инжекция

Когато данните, които един уеб сайт използва, се съхраняват в база от данни, съществува риск от достъп до тези данни чрез манипулиране на заявките. **Подаване на необработени несигурни данни към заявка създава дупка в сигурността**. Пример със скриптовия език PHP, един от най-използваните в уеб програмирането, е показан по-долу.

```
<form method="POST" action="login.php">
  Username: <input type="text" name="username" /><br />
  Password: <input type="password" name="password" /><br />
  <input type="submit" value="Login" />
</form>
```

Това е един опростен формуляр за вход към система. Кодът по-долу показва обработката от страна на сървъра.

```
<?php
 $username = $_POST["username"];
 $password = $_POST["password"];
mysql_query("select * from users where username = '$username' and pass-
word = '$password'");
?>
```

Данните, които сървърът приема през POST заявка директно от потребителя, са незащитени по никакъв начин и се поставят в SQL заявката към базата данни, където се изпълняват. Ако даден потребител въведе `' or ''='` за парола, това ще му осигури достъп за посочения потребител, понеже паролата не се третира като параметър, а като част от заявката. Друг пример би бил въвеждането на `'; Drop table users` за парола, което ще изтрие таблицата `users` от базата данни. Очевидно е, че такива действия не са желани. Можем да се предпазим от това, ако входните данни минават през функция, която да ги обработва, за да не се интерпретират като нищо друго, освен параметър. Пример за такава функция в PHP е `mysql_real_escape_string()`. Доста по-професионален подход е използването на **интерфейс** или **framework**, който да се грижи за тези неща без нужда от действие от страна на програмиста. Такъв интерфейс в PHP е **PDO - PHP Data Objects**. Чрез PDO се създава абстракция между програмния код и

базата данни, където се изпълнява заявката. Примерът по-долу показва използването на така наречените подготвени заявки (Prepared Statements).

```
$sql = “select * from users where username = :username and password =
:password“;
$select = $dbh->prepare($sql);
$select->execute(array(‘:username‘ => $username, ‘:password‘ => $pass-
word));
```

PDO интерфейсът се грижи за подготовка на параметрите, преди подаването им към базата данни. Тази заявка вече е защитена от SQL инжекции. Експертите в програмирането си служат и с други инструменти за достъп до базата данни, които осигуряват желани функционалности в сферата на обектно-ориентираното програмиране, като например **Object Relational Mapping** - представяне на всяко едно поле от таблица от бази данни като обект, **Database Abstraction Layer** - абстракционен слой между приложението и базата данни, който прави до известна степен приложението независимо директно от базата данни, унифицирайки комуникацията. Пример за такива инструменти за PHP са ***Doctrine*** библиотеките. Професионалистите трябва винаги да са насочени към такива библиотеки и интерфейси с оглед не само на сигурността, но и спазването на добрите практики и пестене на ценно време.

Нарушена автентикация и управление на сесиите

Представете си, че резервирате билет за пътуване със самолет. Намирате се на адрес `http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Zurich` и искате да покажете резервацията на някой, като изпращате този линк, който също съдържа ID на сесията. Така потребителят, който отвори този линк, ще използва вашата сесия, която може би съдържа и информация като номер на кредитна карта. Друг пример за проблем с управление на сесиите би било посещение на уеб сайт от **компютър, който е достъпен до много хора**. Ако сесията не се затвори със затваряне на браузъра, следващият потребител, който отвори този уеб сайт, ще има достъп до вашия потребителски акаунт, ако преди това не е натиснат бутонът за изход.

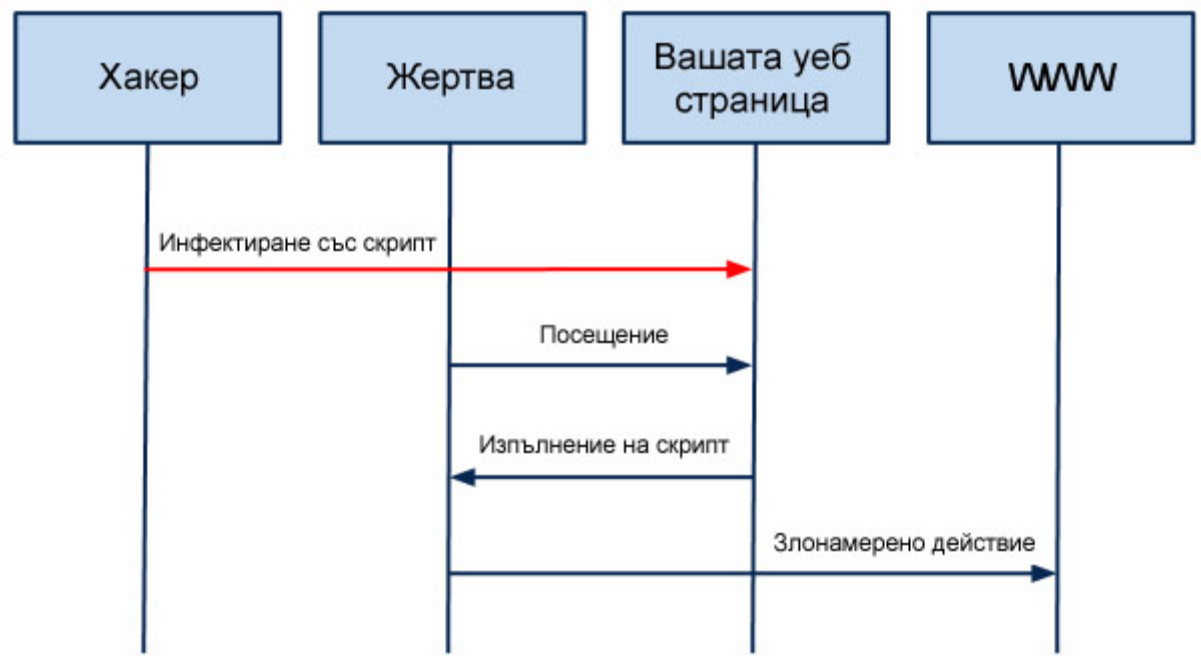
Какво би станало, ако някой злонамерен потребител (дори и да е в рамките на организацията) се сдобие с достъп до паролите на всички потребители? Ако те не са хеширани, този потребител ще има достъп до акаунтите на всички потребители в дадената организация.

Очевидно е, че всички тези описани ситуации са крайно нежелани. Решение на този проблем трябва отново да се постигне на ниво разработка на приложението - управлението на сесиите трябва да отговаря на стандарти за сигурност и паролите на потребителите трябва задължително да се хешират. Пример за такива стандарти са ***OWASP Application Security Vertification Standard Project.***

Cross-Site Scripting (XSS) или крос-сайт скриптинг

Крос-сайт скриптингът (XSS) позволява на нападателите да изпращат злонамерен код към друг потребител, като се възползват от пробойна или слабост в Интернет сайт. Нападателите се възползват от XSS средства, за да инжектират злонамерен код в линк, който изглежда, че заслужава доверие. Хакерът може да вмъкне код в линка на спам съобщение или да използва имейл измама с цел да подмами потребителя да смята, че източникът е легитимен. Когато потребителят щракне върху линка, вграденото програмиране се включва и се изпълнява на браузъра на потребителя. **Нападателите използват XSS, за да се възползват от уязвимостите на машината на жертвата и от зловредния код в трафика, вместо да атакуват самата система**.

Например, нападател може да изпрати на жертвата имейл съобщение с URL, което сочи към уеб сайт и го снабдява със скрипт за влизане, или **публикува злонамерен URL в блог или сайт на социална мрежа като Facebook или Twitter.** Когато потрбителят щракне върху линка, зловредният сайт, както и скриптът, заработват в браузъра му. Браузърът не знае, че скриптът е злонамерен и сляпо изпълнява програмата, която на свой ред позволява на скрипта на нападателя да получи достъп до функционалността на сайта, за да открадне бисквитки или да извърши действия, представящи се за легитимен потребител.



Общ шаблон на XSS атака

Представете си, че посещавате уеб магазин, който взима заглавието от на своя страница(съдържанието между `<title></title>`) от URL - `http://www.examplestore.com/catalog.php?product=25&title=Last%20Chance!`, посредством Java Script:

```
<script>
var url = window.location.href;
var pos = url.indexOf(„title=“) + 6;
var len = url.length;
var title_string = url.substring(pos,len);
document.write(unescape(title_string));
</script>
```

Ако адресът на страницата се промени на `http://www.examplestore.com/catalog.php?product=25&title=<SCRIPT>alert(‘Xss%20Vector!’)</SCRIPT>`. Това ще накара браузъра да покаже малък прозорец.

Само по себе си, това е безвредно, но за съжаление възможностите на тази техника не се изчерпват с показване на `Alert` прозорчета.

Някои общоприети най-добри практики за предотвратяване на кросайт скриптинг включват **тестване на кода на приложението преди внедряване и патчване на пробойните и уязвимостите** по най-бързия начин. Уеб разработчиците трябва да **филтрират въвежданията на потребителите, за да премахнат възможни злонамерени знаци и скриптове** и да инсталират код за **филтриране на потребителските данни**. Администраторите могат също да конфигурират браузърите да приемат само скриптове от доверени сайтове или да изключат скриптовете на браузъра, макар че това може да доведе до уеб сайт с ограничена функционалност.

Колкото времето върви напред, толкова по-добри стават хакерите, използвайки колекция от набори с инструменти, за да ускорят процеса на експлоатиране на уязвимостите. Това означава, че простото внедряване на тези общи XSS практики за защита вече не е достатъчно. Процесът на предотвратяване трябва **да започне по време на разработването**. Уеб приложенията, които са изградени с използване на методология за сигурно разработване на жизнения цикъл, е по-малко вероятно да покажат уязвимости в окончателната версия. Това ще подобри не само сигурността, но също и ефективността и общата цена за притежание, тъй като **адресирането на проблема на живо е по-скъпо, отколкото по време на разработването**.

Неподсигурени директни извиквания на обект

Представете си, че публикувате обява в сайт за продажби по Интернет. След публикуването желаете да я промените и уеб сайтът ви води до този адрес: `http://www.examplesalesite.com/edit_offer.php?id=95732`. Програмната част на сайта приема обявата по този начин:

```
<?php
$offerId = $_GET[‘id‘];
$offerObject = New Offers($offerId);
//access to the object is gained
?>
```

Този код приема ID на обява чрез GET заявка на браузъра и **директно инициализира обекта на тази обява**, без да проверява дали тя принадлежи на потребителя. По този начин потребителят може да **получи достъп до обект, който не принадлежи на него,** и да го манипулира, въвеждайки друго ID в адреса на страницата. За да се избегне този проблем, **уеб приложението трябва да проверява дали потребителят има право на достъп до обекта**, който е поискал:

```
<?php
$currentUser = Users::getCurrentUser();
$offerId = $_GET[‘id‘];
if (Offers::offerBelongsToUser($offerId, $currentUser->getId) === false)
    die(“You are not authorized to view this page.“);
$offerObject = New Offers($offerId);
?>
```

Този код извиква обекта на потребителя, който разглежда страницата, обръщайки се към статичен метод в клас „потребители“, след което **проверява дали е безопасно да се даде достъп на този потребител до поискания обект** чрез съпоставяне на обявата и потребителя (извикване на статичния метод в клас „Обяви“ `boolean offerBelongsToUser()`). Ако методът върне стойност „лъжа“, то тогава потребителят не трябва да получава достъп до поискания от него обект - вероятна атака.

Неправилно конфигуриране, засягащо сигурността

Няколко примери за това:

- *Конзолата за управление на сървъра е автоматично инсталирана и след това не е изтрита. Данните за вход, които са по подразбиране, не са премахнати. Злонамерен потребител открива, че стандартните страници за администриране са на сървъра и получава достъп чрез използване на потребител и парола по подразбиране.*
- *Показване на списъка с файлове в дадена папка не е забранен (**Directory Listing**). Злонамерен потребител намира и изтегля всички компилирани Java класове, които се декомпилират, и **открива сериозни пропуски в сигурността на Java приложе-***

ието.

- *Сървърът е конфигуриран така, че да **показва подробна информация за настъпили грешки** в рамките на приложението. Хакерите обожават допълнителната информация, която съобщенията за грешки предоставят.*
- *Сървърът съдържа примерни приложения, които не са изтрити и за които се знае, че имат сериозни пропуски в сигурността.*

Няколко отправни точки биха помогнали за адресирането на проблеми от такова естество: ***надеждна архитектура на приложението***, която предоставя ефективно и сигурно разграничение на различните компоненти, извършване на периодични сканирания и одити, които да помагат при откриването на проблеми в конфигурацията и **подновяване на използван софтуер на време**.

Излагане на чувствителни данни

Първият въпрос, който трябва да се запитаме, е **кои данни са чувствителни и имат нужда на защита**? Това са пароли, номера на кредитни и дебитни карти, лична информация и др. За всички тези данни:

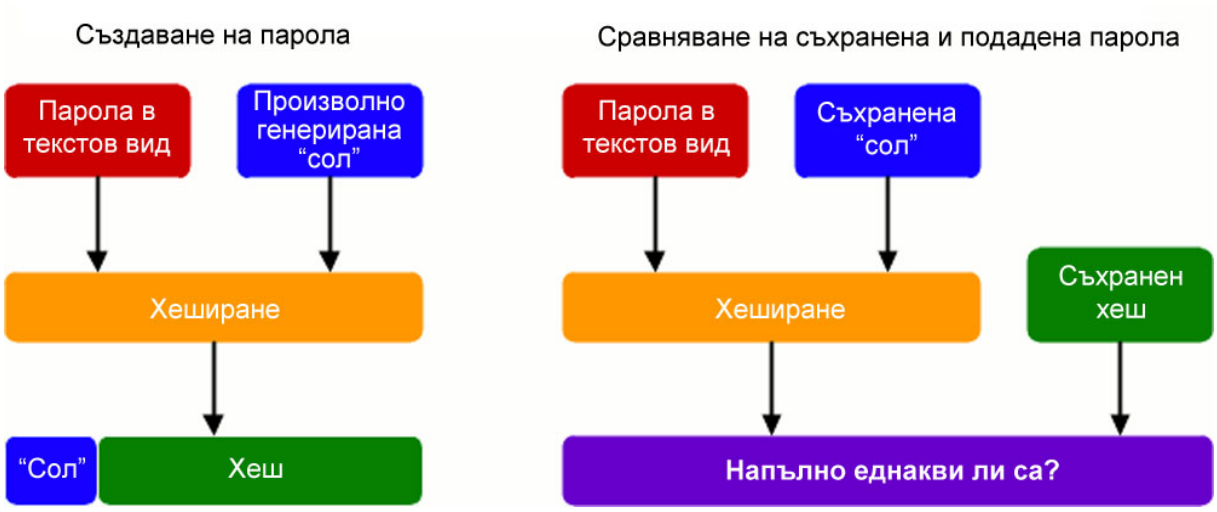
- *Съхраняват ли се някъде в **чистия си текстов вид** за дълго време (вкл. back-ups)?*
- *Прехвърлят ли се вътрешно или външно в чистия си текстов вид?*
- *Използват ли се **слаби или стари алгоритми за криптиране**?*
- *Правилно ли са зададени header fields на HTTP протокола за заявки и отговори (header fields - компоненти на заглавната част на съобщението, като например контрол на кеширането, бисквитките и много др.)?*

Най-малкото, което трябва да се направи, за да се предпазят чувствителните данни, е:

- ***Определете заплахите**, от които ще предпазвате чувствителните данни - вътрешна атака, външен потребител. Криптирайте както данните, които се съхраняват, така и данните, кои-*

то се предават.

- ***Не съхранявайте чувствителни данни, които не ви са необходими**. Няма как да бъдат откраднати данни, които не притежавате.*
- *Използвайте **силни алгоритми и силни ключове** за криптиране и управлявайте сигурно ключовете.*
- *Криптирайте съхранените пароли с алгоритми, **специално създадени за защита на пароли**.*
- ***Забранявайте кеширането на страници**, които съдържат чувствителна информация.*



*Схема, демонстрираща сигурно създаване и съпоставяне на съхранена и подадена от потребителя парола. **Сол** в контекста на криптографията е произволно създадена комбинация от знаци, която се добавя към хеширащи функции за допълнителна защита на пароли.*

Съхраняването на номера на кредитни и дебитни карти прави вашия уеб сайт желана мишена за хакери. Предвид юридическата отговорност и възможните последици от достъп на хакер до база данни с номера на кредитни и дебитни карти, се препоръчва **тези данни да се съхраняват от трети страни**, например световен лидер в онлайн разплащанията като *PayPal* и *Authorize.Net*.

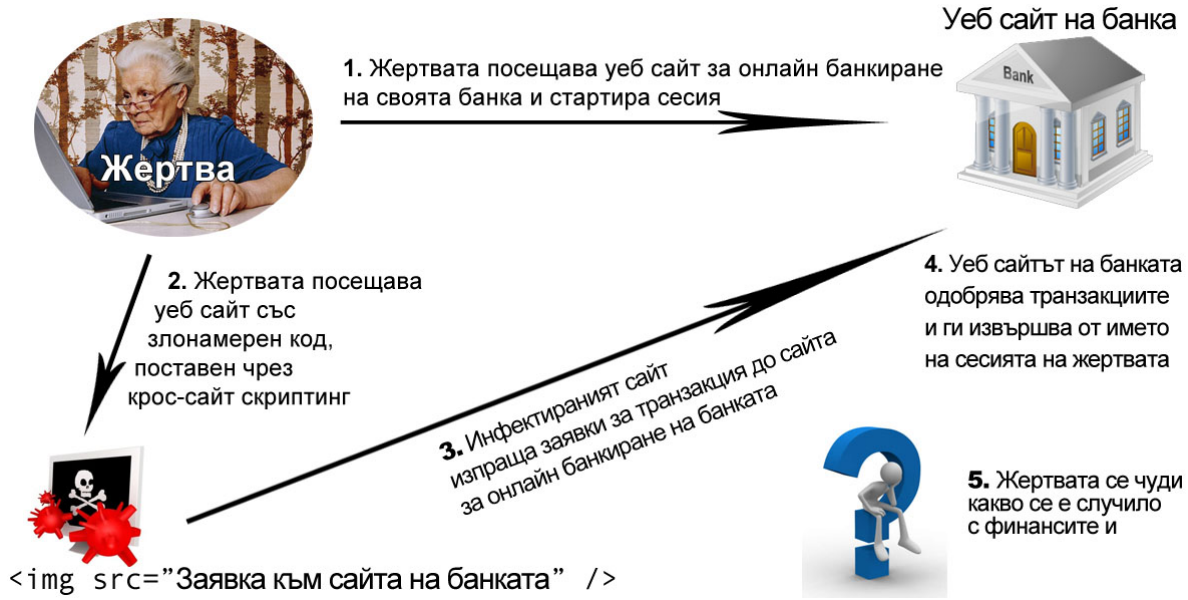
Липса на котрол на достъпа

Уеб разработчиците трябва успешно да ограничават достъпа до страници, които предлагат функционалност, предвидена само за оторизирани потребители. Интерфейсът не трябва да показва линкове към страници, които не трябва да се посещават от неоторизирани потребители, но това само по себе си не е достатъчно. **Не трябва да се пропускат и проверките в рамките на програмния код, които да проверяват дали поисканата страница може да се покаже на потребителя**.

Представете си, че се намирате в уеб форум и сте отворили профила на даден потребител: `http://www.example.com/forum/view_user.php?id=6192`. В рамките на приложението съществува страница, която изтрива даден потребител при подадено ID - `http://www.example.com/forum/delete_user.php?id=6192`. Вие сте обикновен потребител и не виждате препратка към тази страница, но ако успеете да я отгатнете и тя не е защитена от неоторизиран достъп, потребителят с посоченото ID ще бъде изтрит. Очевидно това е пропуск в сигурността.

Извършване на крос-сайт заявки чрез преправяне (CSRF, Cross-Site Request Forgery)

Този вид атаки разчитат на това потребител да е отворил сесия в някакъв сайт и междувременно да посети друг сайт, който да използва сесията на този потребител.



Пример за това е уеб сайт за разплащания, който има страница като тази: `http://www.examplepaymentsite.com/transfer_funds.php?amount=5000&transferToAccount=9343917` - тази страница изпраща сума на посочения потребителски акаунт в GET заявка. Атаката се състои в това потребителят да посети друг сайт, който изпраща GET заявка от името на атакувания потребител и в тази GET заявка да се намира потребителски акаунт, който да получи определената сума. Пример за изпращане на такава заявка е този код:

```
<img src="http://www.examplepaymentsite.com/transfer_funds.php?amount=5000&transferToAccount=attackersAccountNumber" width="0" height="0" />
```

Този проблем се адресира чрез добавянето на произволно създаден ключ или токен (token), който да се включва към URL адреса като GET параметър или през формуляр в скрито поле като POST параметър. Този токен трябва да е различен поне всяка нова сесия. **Страниците, които получават заявки, проверяват дали подадения токен съвпада с генерирания за конкретната потребителска сесия и ако не съвпада, потенциалната атака ще бъде осуетена**. Адресът за изпращане на сума на друг потребител би изглеждал по този начин: `http://www.examplepayment-site.com/transfer_funds.php?amount=5000&transferToAccount=9343917&token=KDIOSJ492DJQK92349CXZ97`.

Използване на софтуерни компоненти, за които се знае, че имат уязвимости

За този проблем съществува решение, което е радикално и крайно нереалистично - използвайте само софтуер, който е написан от вас или вашия екип. **Непродуктивно е да не се използват съществуващите днешно време безплатни, *open source* и платени софтуерни компоненти**. За да се избегне проблемът със съществуващи уязвимости се препоръчва обновяване на съответните компоненти на време и следене на новите версии, които се появяват от разработчиците.

Непроверени препращания

Ако даден уеб сайт има страница, чиято единствена цел е да препрати потребителя към адрес, който получава чрез GET заявка, то тази стра-

ница би могла да бъде използвана за ***фишинг***: `http://www.examplebanksite.com/redirect.php?url=home.php`. Ако параметърът URL не се проверява дали ще отведе някъде, където не трябва, то тогава на този параметър може да се подаде всякаква стойност и целият адрес ще има заблуждаващ вид на сигурен уеб сайт, предвид домейна (`www.examplebanksite.com`). Нищо не подозиращ потребител може да получи е-поща, която да изглежда като изпратена от официалната организация, но която да съдържа линк, който да води към сайт-копие, където да му бъдат поискани данните за вход към системата на организацията: `http://www.examplebanksite.com/redirect.php?url=www.examplebankfakesite.com/login.php`.

Подобен пример е ако злонамерен потребител реши да отгатне страница за администриране на уеб сайта и промени параметъра на този адрес: `http://www.examplebanksite.com/login.php?return=account.php` с този: `http://www.examplebanksite.com/login.php?return=admin.php`. Това може да даде достъп до интерфейс за администриране, ако не съществуват необходимите проверки.

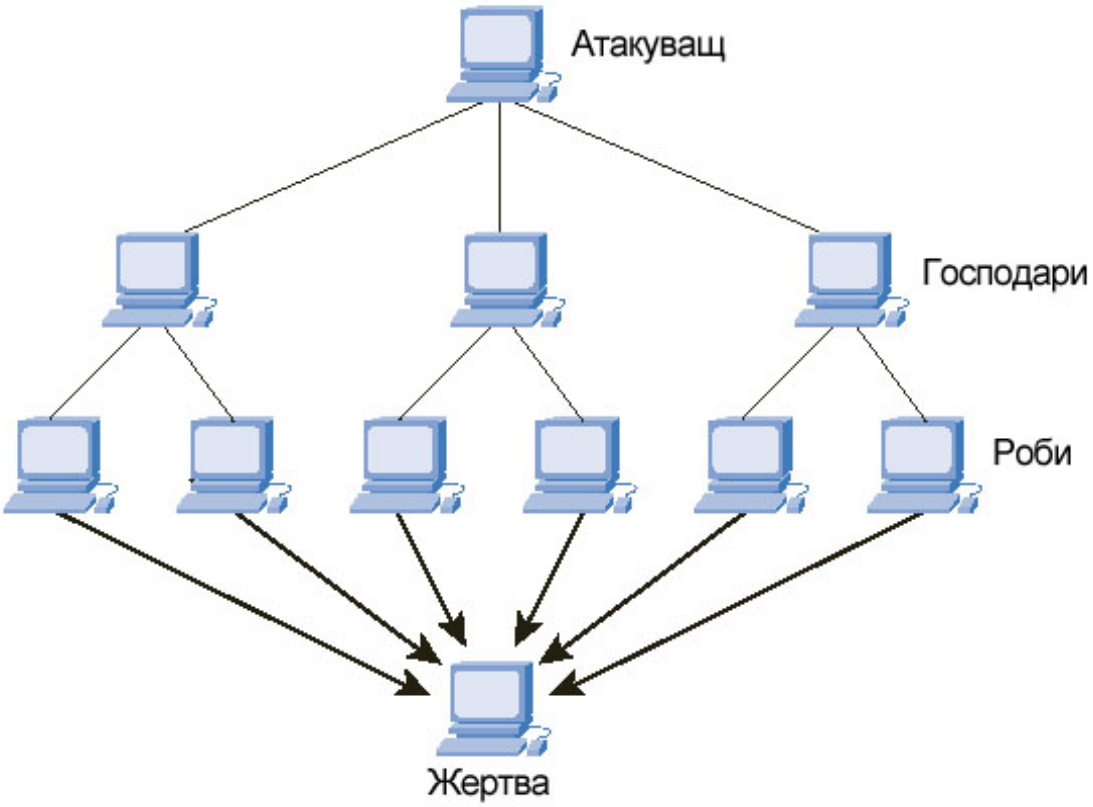
Разпределен отказ от обслужване (DoS - Denial-of-Service)

Още позната като ***DDoS (Distributed Denial-of-Service)***, тази атака може да бъде пагубна за една организация, струвайки й време и пари, като по същество изключи корпоративните системи.

Хакерите извършват DDoS атака, като експлоатират пролуки и уязвимости в компютърната система (често уеб сайт или уеб сървър), за да се позиционират като главна система. След като веднъж са се поставили в положение на главна система, хакерите могат да идентифицират и комуникират с другите системи за по нататъшно компрометиране.

След като нарушителят е поел контрол над множество компрометирани системи, той може да възложи на машините да започнат някоя от многото **атаки за препълване, докато целевата система се препълни с фалшиви искания за трафик**, което ще доведе до **отказ на услуга за ползвателите на тази система**. Потокът от входящите съобщения от компрометираните системи, ще причини спиране на системата цел и отказ от услуги към нея, което води до **невъзможност за достъп на потребители-**

те до каквото и да било, и следователно ще струва на организацията време и пари.



Типична DDoS атака: *машините на* ***Господари*** *и* ***Роби*** *са компрометирани и са инфектирани със злонамерен код. Атакуващият управлява и координира* ***Господарите****, които координират и активират* ***Робите****, които изпращат огромен обем пакети към* ***Жертвата****, наводнявайки системата с безполезен товар, който изтощава наличния ресурс.*

Предотвратяването на DDoS атака може да бъде трудно, тъй като тя е предизвикателство за това как да се направи **разграничение между зловредна заявка за трафик и легитимна такава**, тъй като те използват еднакви протоколи и портове. Все пак има няколко стъпки, които могат да се предприемат за защита на вашите системи от разпределени атаки, предизвикващи отказ от услуги:

- ***Уверете се, че имате излишък от честотна лента във връзката на организацията с Интернет****: Това е една от най-лесните защити срещу DDoS, но може да се окаже доста скъпа. Просто като имате многочестотна лента за обслужване на заявките за трафик, това може да помогне за предпазване от ниско ниво DDoS атаки. Също така, колкото по-широка честотна лен-*

та има организацията, толкова повече трябва да направи нападателят, за да запуши връзката й.

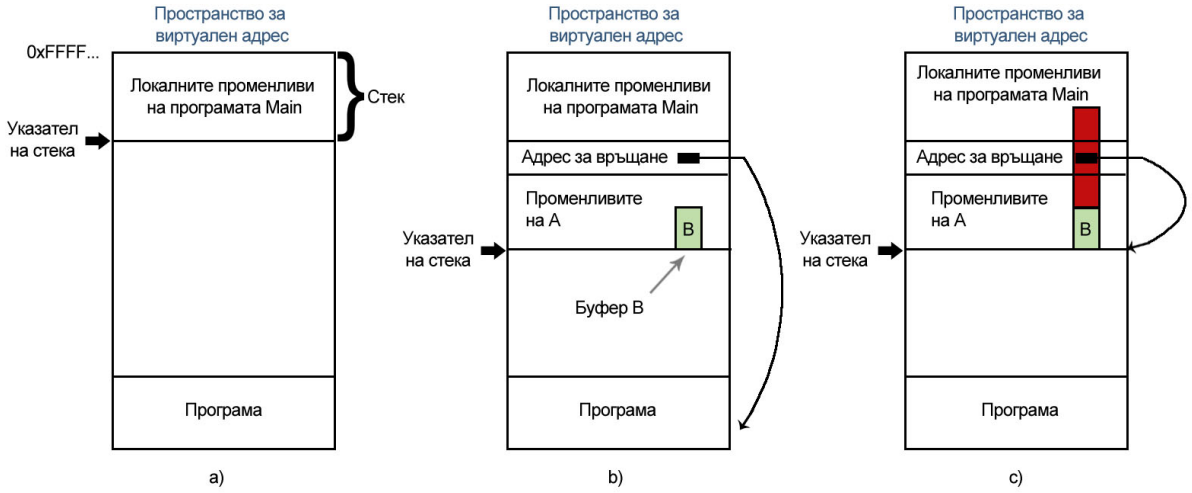
- ***Уверете се, че използвате система за откриване на проникванията (Intrusion Detection System, IDS)**. Няколко от наличните днес системи за откриване на проникванията са снабдени с технологии за защита на системите от DDoS атаки, като използват методи за проверка и потвърждаване на връзката и за предотвратяване достигането до корпоративните сървъри на определени заявки.*
- ***Използвайте продукт за защита от DDoS.** Няколко производители предлагат устройства за DDoS защита и предотвратяване, които са конструирани специално за откриване и осуетяване на DDoS атаки.*
- *Използването на регулиращи и ограничаващи технологии може да намали въздействието от DDoS атаката.*
- ***Поддържайте резервна Интернет връзка с отделна база с Интернет адреси за критични потребители.** Това ще ви предложи алтернативен път, ако първичната верига е претоварена със злонамерени заявки.*

Препълване на буфер (buffer overflow)

Буферът е временна област за съхранение на данни. Когато там се поставят повече данни, отколкото е предвидено първоначално от даден програмен и системен процес, допълнителните данни ще го препълнят, откъдето произлиза и името, **причинявайки част от данните да изтекат към други буфери**, което може да разруши данните, които те съдържат или да запише върху тях.

При атака за препълване на буферите препълващите данни понякога съдържат специфични инструкции за дейности, проектирани от хакери или злонамерени потребители. Например **данните могат да превключат отговор, който уврежда файловете, променя данните или разкрива частна информация**.

Хакерите биха използвали буферното препълване, за да се възползват от програма, която очаква въвеждане от потребителя. **Има два типа буферно препълване – стек базирани и хип базирани**. Хип (от heap – купчина) базираните са по-трудни за изпълнение и затова се срещат по-рядко. Те атакуват приложението чрез „наводняване“ на пространството памет, запазено за програмата. **Стек (от stack – купчина) базираното препълване на буфера, което се среща по-често сред хакерите**, експлоатира приложения или програми, като използва това, което е известно като стек – пространство от паметта, използвано за съхранение на потребителски въвеждания.



*На **фигура a)** виждаме работеща главна (Main) програма, заедно с нейните променливи в стека. Тя извиква **процедурата A**, както се вижда на **фигура b)**. Стандартната последователност на извикване започва с изместването на **адреса за връщане в стека**. След това прехвърля контрол на A, която **сваля указателя на стека, за да разпредели памет за собствените си променливи**. Да речем, че **буферът на A е фиксиран на 1024 байта**. Ако потребителят на програмата подаде данни, по-големи от 1024 байта, се случва **препълване на буфера** и данните се написват върху паметта на друг процес, както е показано на **фиг. c)** в червено. Ако данните са достатъчно големи, те надписват и адреса за връщане. Ако **буферът съдържа злонамерена програма** и адресът за връщане е променен, когато процедурата A приключи, програмата в B ще започне да се изпълнява. По този начин **злонамерен потребител може да вмъкне код в програмата и да го изпълни**.*

Един стек може да поддържа само определено количество данни и ако входният низ е по-дълъг от резервираното пространство, резултатът е **препълване, създаващо дупка в сигурността**. Хакерите търсят такива

пробойни със специално написани команди, които причиняват препълване и задействат атаката. След като злонамерената команда е причинила препълване, хакерът все още трябва да изпълни командата, като посочи адрес за връщане, който сочи към командата. **Препълването на буфера „счупва“ частично приложението, но то се опитва да се възстанови, като отива към адреса за връщане**, който е бил пренасочен към злонамерената команда от хакера.

Когато атаката с препълване на буфера задейства командата, намерена в новия адрес за връщане, програмата смята, че все още работи. Това означава, че командният прозорец, който е бил отворен, работи със същия набор изпълними разрешения, както приложението, което е било компрометирано, позволявайки на хакера да получи пълен контрол над операционната система.

Мерките за справяне с този вид атаки са както следва:

- ***Избягвайте да използвате библиотечни файлове.*** *Библиотечните файлове, които се използват в езиците за програмиране и по природа са несигурни, са цел за хакерите по време на атаките срещу приложенията. Всяка слабост, намерена от хакера в библиотечния файл, ще съществува също във всички приложения, които използват библиотечни файлове, давайки на хакерите блестяща цел за потенциална атака.*
- ***Филтрирайте вероятни опасни HTML кодове и знаци, които могат да причинят проблеми с базата данни*** *(в някои езици за програмиране това са апострофът, кавичките, амперсантът).*
- ***Тествайте приложенията преди внедряването им.*** *Опитайте се да пробиете всяко приложение, за да си осигурите сигурни кодове.*

Заключение

Развиващите бизнес в Интернет, дори и да са практически некомпетентни в разработването на уеб приложения, трябва да изискват от своя

екип разработчици или организацията, с която работят, **утвърждаване и редовно използване на добри практики за защита на приложенията от вече описаните и други потенциални атаки**. Трябва **да се наблегне на проблемите по сигурността при обучение на професионалисти** в областта на разработването на софтуер и същите трябва редовно да следят **тенденциите в индустрията**, поява на нови видове атаки и поява на нови и иновативни методи на защита.

Създаването на сигурно уеб приложение или подсигуряването на вече съществуващо такова е трудна задача. ***OWASP*** (*The Open Web Application Security Project - Отвореният проект за сигурност на уеб приложенията*) помага на организации и уеб разработчици, като предлага **безплатни ресурси, които адресират рисковете за сигурността на уеб приложенията**.

Сигурност на уеб браузърите

Браузърите често биват използвани за злонамерени атаки чрез Java Script и/или Flash. **Хакери се възползват и от някои уязвимости, характерни за всички браузъри.** Сигурността на браузърите може да бъде пробита по няколко начина:

- *Операционната система е компрометирана от зловреден код, който прочита/променя с приоритет паметта, която използва браузъра*
- *Изпълняващият браузъра файл може да е компрометиран*
- *Добавките към браузъра (addons, plug-ins) може да са компрометирани*
- *Комуникацията на браузъра може да се прихване извън машината*

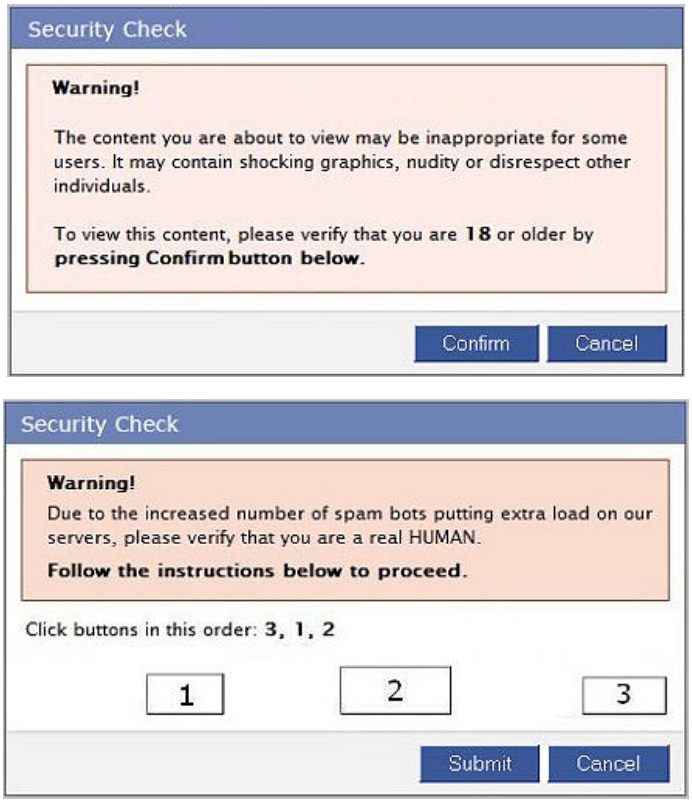
Уеб браузърът е вероятно да не осъзнава която и да е от тези ситуации и да показва на потребителя, че връзката е сигурна.

Причината за пробиване на сигурността на браузърите е често с цел заобикаляне на защитата от поп-ъп прозорци и събиране на лична информация с цел кражба на личност или маркетинг. Някои похвати, с които зло-

намерени потребители и организации си служат, са:

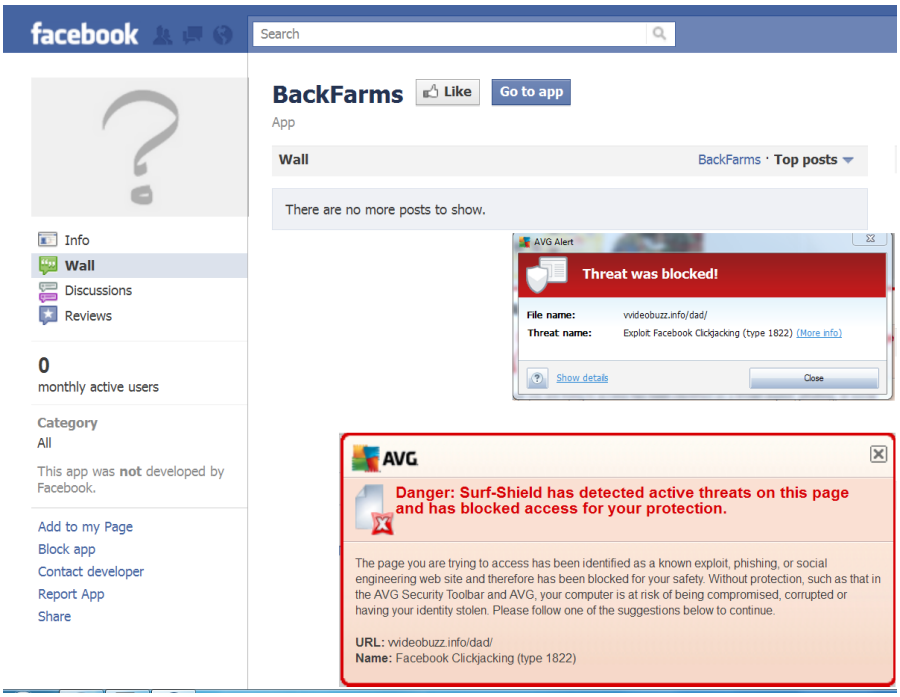
- ***Кликджекинг (Clickjacking)*** *- техника, при която уеб потребител е заблуждаван с това, че отваря хиперлинк, който води до съдържание, което потребителят не е очаквал и не отговаря на това, което е приел първоначално.*
- ***Лайкджекинг (Likejacking)*** *- техника, която се отнася към Facebook - потребител на Фейсбук „харесва" страница, която не е желал умишлено да „хареса". Това води до публикуване на съдържание на неговата лична Фейсбук страница.*

Уязвимостите на уеб браузърите могат да бъдат намалени чрез често обновяване на тях. Статистиката показва, че с времето се появяват все повече и повече уязвимости в браузърите, но разработчиците им се отнасят по-сериозно към тях и реагират по-бързо на открити такива, чрез издаване на нови версии. Но ако операционната система е заразена с зло-



Тези два прозореца се представят като проверка за сигурност - единият изисква потребителят да потвърди, че е навършил 18год., а другият да подреди числата 1,2 и 3, за да докаже, че е действително човек, а не машина. Реално тези прозорци публикуват съдържание на страницата на потребителя във Фейсбук и нямат нищо общо с проверките, за които се представят.

вредна програма, това няма да е достатъчно. Някой подкомпоненти на уеб браузърите са особено уязвими, като например бисквитките и допълнителните приложения на браузъра (add-ons).

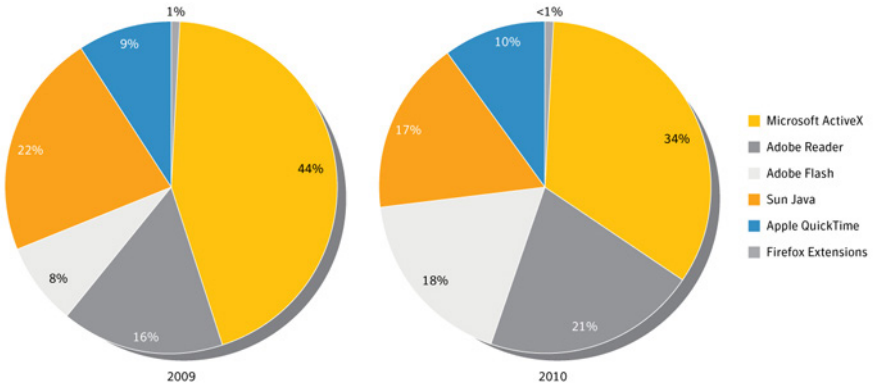


Антивирусна програма блокира Лайкджекинг атака

Разширения и плъг-ини на браузъра

Въпреки че разширенията не са сами по себе си част от браузъра, те **увеличават полето за изява на хакерите**, като откриват уязвимости в ***Abode Flash Player, Adobe Acrobat Reader, Java Plugin, ActiveX***. Зловреден код може да се помести в разширение на браъузъра; някои браузъри, като Google Chrome и Mozilla Firefox блокират и предупреждават потребителя за несигурни разширения.

Adobe Flash също съхранява бисквитки на машината на потребителя. Повечето функции на браузърите за триене на кеша и историята не влияят на кеша на Flash. **За това потребителите, които са изчистили данните**, които уеб сайтовете използват за следене на потребителите (бисквитки, сесии), **може да вярват, че са премахнали всичко, но всъщност историята на Flash е все още непокътната**.



*Статистика на **Symantec Corporation** за уязвимостите на плъг-ините на браузърите, 2009-2010г.*

Представената графика по-горе обхваща общия брой уязвимости, документирани от упомената организация. Включени са и потвърдени, и непотвърдени уязвимости. **Потвърдените уязвимости са тези, за които публично е заявено от създалите плъг-ина компании**, че съществуват. Непотвърдени уязвимости са тези, които са открити от трети страни, но за които съответната компания не е издала публично потвърждение. **Ако уязвимостта е непотвърдена не значи, че не е основателна и допустима.**

Symantec Corporation е направила анализ на следните системи:

- *Adobe Reader*
- *Adobe Flash Player*
- *Apple QuickTime*
- *Microsoft ActiveX*
- *Mozilla Firefox extensions*
- *Oracle Sun Java platform Standard Edition (Java SE)*

Уязвимостите на плъг-ините са се увеличили за 2010-та година. За 2009 документираните от Symantec са 302, докато за следващата година те нарастват до 346.

Уязвимостите на ActiveX намалят. ActiveX е технология, разработе-

на за Internet Explorer, който е браузър с прогресивно намаляваща популярност. Освен това, Internet Explorer 8 въвежда подобрения в сигурността на технологията.

Продуките на Adobe са все повече и повече мишена за хакери. Adobe Reader и Flash се използват от хакери като вектор за разпространение на зловреден софтуер към произволни нищо не подозиращи потребители и точно определени такива.

Браузъри за мобилни устройства

Все повече и повече хора сърфират в Интернет чрез мобилните си устройства - телефони, таблети и т.н. **Мобилните браузъри не са толкова сигурни, колкото браузърите на компютрите.** Хората са по-уязвими в Интернет, когато сърфират от мобилно устройство, в сравнение с компютър. **Антивирусните програми за таблети и мобилни телефони са все още на много ниско ниво и са в процес на разработка.** Това прави мобилните устройства съблазнителна и примамлива цел за хакери.

Социално инженерство

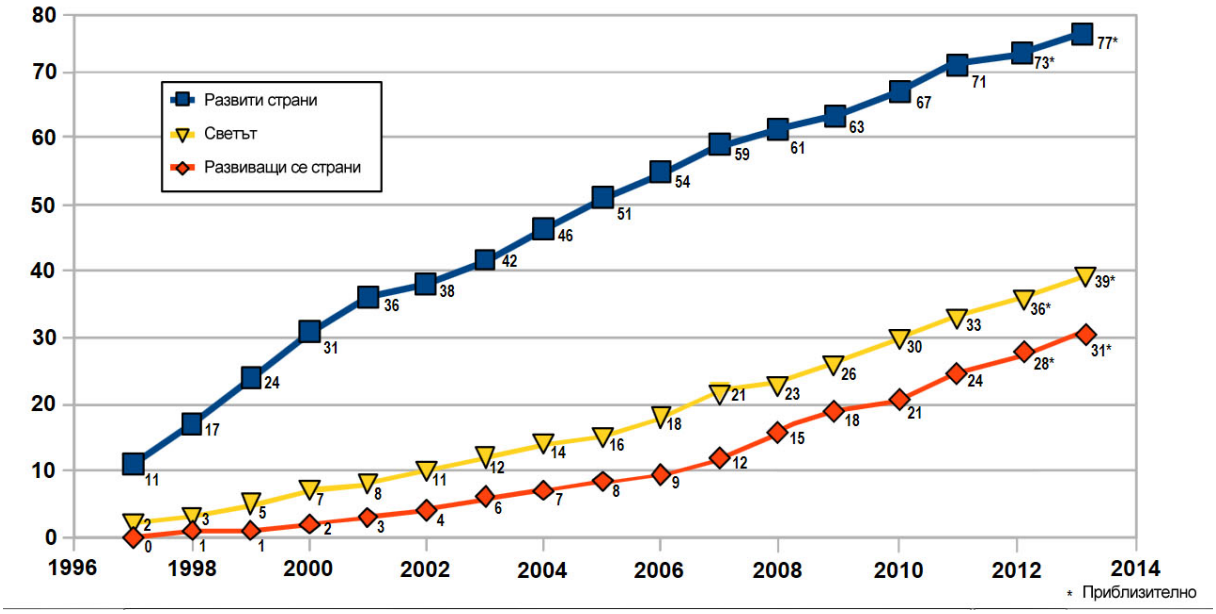
Основната цел на социалното инженерство е същата, като на хакерството: **придобиването на неоторизиран достъп до системи или информация, с цел измама, кражба на идентичност, индустриален шпионаж, или причиняване на друга вреда**.

Основната разлика е, че **социалното инженерство използва главно психологически методи**, а именно естествената за човека склонност да се доверява *(дори за умели хакери много често е по-лесно да пробият сигурността и да получат желаната информация, като просто вдигнат телефона и попитат счетоводителя за паролата му, вместо да използват техническите си умения)*.

Какво прави социалното инженерство толкова ефективно? Психологическият подход, използващ утвърдени методи за убеждаване: представяне за някой друг, конформизъм, позоваване на авторитетна фигура, разсейване на отговорността или просто дружелюбно отношение.

Заключение

С нарастващата зависимост на бизнеса от информационните технологии и уеб пространството и увеличаващото се потребление на уеб услуги **нараства и интереса на хакерите към компрометиране чрез злонамерени действия**. Причините са много и варират - от чисто удоволствие, през нужда за публично внимание и завършват с бизнес - финансово благополучие чрез измама или атака на конкуренцията.



Интернет потребители на 100 жители от 1997-ма година насам

Проблемът съществува и се задълбочава. Появяват се нови и по-сложни уеб технологии, заедно с които се развиват разнообразни и креативни начини за компрометиране. **Организациите инвестират все повече и повече време и ресурси в насока подобряване на сигурността, но постигане на сигурност в уеб пространството е нещо комплексно.** Усилия трябва да се положат и от предлагащите уеб услуги, и от използващите ги.

В потребителите е нужно да се създаде култура на сърфиране в Интернет. Те трябва да са наясно с това, че Интернет не е безопасно място. Остава въпросът: ***по какъв начин е възможно практически ефективно да се разпространи този факт и да се представи по такъв начин, че да привлече вниманието на потребителя и да го накара да***

поеме отговорност за своето присъствие в мрежата? Всеки един уеб потребител трябва да знае как да създаде сигурно своята парола и как да я пази, как да разпознава какви хиперлинкове е безопасно да отвори и какви - не, кои писма в електронната си поща да игнорира и т.н. **Предвид достъпността на Интернет, това се превръща в утопия за бранша**. Съществува друг подход, който има потенциал да даде много по-значим ефект - обучение на добри и компетентни кадри в областта на софтуерното разработване, където да се набляга върху създаване на софтуер с минимални уязвимости и акцентира върху изключителната и непренебрежима важност на сигурността на приложенията. На теория това трябва да се отрази върху софтуера, който е достъпен до потребителите в Интернет, в аспект **защита на информацията**.

Използвана литература

http://en.wikipedia.org/wiki/Message_authentication_code

http://en.wikipedia.org/wiki/Transport_Layer_Security

http://www.networkworld.com/news/2007/011807-tls2.html

https://securityblog.redhat.com/2013/07/24/transport-layer-security/

http://www.digicert.com/ssl.htm

http://bg.wikipedia.org/wiki/SSL

http://publib.boulder.ibm.com/tividd/td/ITLM/SC32-1431-01/en_US/HTML/tlminmst45.htm

http://validationcertificateinfo.blogspot.com/2012/12/the-advantages-and-disadvantages-of.html

http://publib.boulder.ibm.com/tividd/td/ITLM/SC32-1431-01/en_US/HTML/tlminmst45.htm

http://cryptography-help.hit.bg/symmetric_cryptography.html

http://cryptography-help.hit.bg/asymmetric_cryptography.html

http://teacher.bg/cs/blogs/avatara/archive/2009/07/30/27838.aspx

http://en.wikipedia.org/wiki/Global_Internet_usage

http://www.esecurityplanet.com/trends/article.php/3933796/Uncovering-the-Top-3-Browser-Vulnerabilities.htm

http://www.symantec.com/threatreport/topic.jsp?id=vulnerability_trends&aid=browser_plug_in_vulnerabilities

http://elearningbg.wordpress.com/2010/09/09/%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D0%BB%D0%BD%D0%BE-%D0%B8%D0%BD%D0%B6%D0%B5%D0%BD%D0%B5%D1%80%D1%81%D1%82%D0%B2%D0%BE/

http://en.wikipedia.org/wiki/Browser_security

http://review.sagabg.net/kak-da-zashitim-ueb-prilozheniyata-predotvratyavan.html

https://www.owasp.org/index.php/Main_Page

http://www.acunetix.com/websitesecurity/cross-site-scripting/

http://www.chmag.in/article/aug2010/advance-xss-attacks-dom-based

http://packetlife.net/blog/2011/feb/21/intelligent-password-storage/

http://www.eset.com/bg/home/products/antivirus/anti-phishing/

http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_7-4/dos_attacks.html

http://www.read.seas.harvard.edu/~kohler/class/05f-osp/notes/lec19.html