PROTÓTIPO DE UM LABORATÓRIO DE SEGURANÇA

ATAQUES CRIPTOGRÁFICOS E DE SQL INJECTION

MARIA FILIPA MARTINS MOTA DA SILVEIRA E CASTRO CRUZ GOMES

Dissertação para obtenção do grau de Mestre em Segurança de Informação e Direito no Ciberespaço

Orientador: **Pedro Miguel dos Santos Alves Madeira Adão**

Júri

Presidente: **Paulo Alexandre Carreira Mateus**

Vogal: **Nuno Miguel Carvalho dos Santos**

Vogal: **Pedro Miguel dos Santos Alves Madeira Adão**

Outubro de 2017

Agradecimentos

Obrigada ao Miguel e aos nossos filhos, Tomás, Bernardo, Vera e Vasco, por todo o apoio e incentivo, cada um à sua maneira.

Obrigada à Mafalda Rebelo Pinto, à Inês Cordeiro de Sousa e à Margarida Martins Mota.

Obrigada ao Prof. Pedro Adão pela dedicação e ajuda incansáveis.

Resumo

A evolução tecnológica das últimas décadas facilitou o aparecimento de novas formas de comunicação. Este novo modo de interacção digital trouxe enormes vantagens, permitindo agilizar a relação de todo o tipo de comércio e serviços com os seus clientes. Porém, para que todas estas actividades possam ser desenvolvidas e adoptadas, é necessário que haja confiança na sua utilização, ou por outras palavras é fundamental que a confidencialidade, a autenticidade e a integridade das comunicações sejam asseguradas. A criptografia é uma arma poderosa para alcançar esses objectivos. Esta ciência antiga tem-se adaptado aos tempos modernos e têm sido desenvolvidas cifras fortes, com algoritmos de cifra que em alguns casos são muito difíceis de reverter sem o conhecimento da chave. No entanto, em determinadas circunstâncias particulares, uma escolha de parâmetros pouco criteriosa pode fragilizar a solidez da segurança de sistemas criptográficos tão robustos como o RSA. Os ataques de Wiener, de Common-modulus são disso um bom exemplo. O Least Significant bit (LSB) é um exemplo de um ataque à implementação criptográfica.

A segurança das comunicações pode ser colocada em causa de muitas outras formas. A Informação é um bem valioso e muito desejado, e a procura de falhas que levem à sua revelação é uma área em constante evolução. Hoje, a comunicação com as mais diversas entidades é muitas vezes realizada através de páginas web que interagem com os utilizadores, recebendo e devolvendo informação. Estes *sites*, em geral, usam bases de dados para organizar e gerir os seus conteúdos. Em alguns casos a divulgação da informação é promovida e incentivada, como no caso do marketing, mas em outros, pelas mais variadas razões, nomeadamente por razões legais ou de segurança, o seu alcance deve ser vedado, estando apenas acessível a alguém autorizado. As SQL Injections são uma forma eficaz de ludibriar a máquina, obrigando-a a revelar informações. Esta técnica usa os campos de diálogo (autenticação, pesquisa, …) das páginas web para introduzir o ataque e induzir a máquina a revelar, manipular ou até mesmo a destruir a informação.

A segurança da informação está longe de poder ser reduzida a um mero problema informático. A aliança de esforços desta área com a Segurança de Informação nas Organizações e o Direito são a base para potenciar uma segurança mais eficaz e minimizar as consequências dos incidentes, mesmo sabendo que o objectivo de tornar a segurança infalível é de difícil alcance.

A formação é um factor que muito contribui o correcto *design* e utilização segura destes sistemas, quer na vertente teórica, na divulgação dos vários conceitos, quer também na vertente prática, através de ambientes de teste onde é possível experimentar e verificar o alcance dos erros, sem consequências reais.

No âmbito desta tese foi desenvolvido um protótipo de um laboratório de segurança para demonstrar os ataques criptográficos de Wiener, Common-modulus e Least Significant Bit, e testar os ataques de SQL Injection. Foram construídos servidores que implementam

erradamente cifras numa máquina virtual, i.e., que geram os repectivos parâmetros negligenciando as fragilidades que foram detectadas nos ataques ao RSA acima descritos e foram também desenvolvidos os algoritmos que tornam possível a decifra dessas mensagens geradas aleatoriamente. Foi também desenvolvida, numa outra máquina virtual, uma aplicação web que está ligada a uma base de dados, cujo acesso é feito através da introdução de dados de autenticação utilizando o nome do utilizador e respectiva *password.* Esta aplicação é vulnerável e permite ludibriar este típico controlo de acesso, possibilitando a revelação de informação contida nessa base de dados. Por último foi concebida uma máquina idêntica à anterior, mas neste caso, foi colocada uma *firewall* aplicacional para a web (Mod Security), que analisa a introdução de dados, detecta as SQL injections e bloqueia este tipo de ataques.

Estas máquinas foram pensadas para utilizadores que dão os primeiros passos na área da Segurança da Informação, no entanto, permitem o desenvolvimento futuro de outros módulos de treino, para experimentação de outros tipos de vulnerabilidades.

Palavras-chave: **Criptografia, RSA, Wiener, Common-modulus, LSB, SQLi, Vulnerabilidades, Protótipo laboratório de segurança, Ambiente de testes.**

Este artigo foi escrito de acordo com a antiga ortografia

Abstract

The technological evolution of the last decades has helped the emergence of new forms of communication. This new mode of digital interaction has brought enormous advantages, allowing to streamline the relationship between all types of commerce and services and its clients. However, for all these activities to be developed and adopted it is necessary to trust it, or in other words it is essential that confidentiality, authenticity and integrity are ensured. Encryption is a powerful weapon to achieve these objectives. This ancient science has adapted to modern times and developed strong ciphers with decipher algorithms that, in some cases, are very difficult to reverse without the knowledge of the key. However, in some particular circumstances, a wrong choice of parameters may weaken the security robustness of cryptographic systems such as RSA. The Wiener, Common-modulus attacks are good examples of this flow. The Least Significant bit (LSB) is an example of an attack on the cryptographic implementation.

The security of communication can be compromised in many other ways. Information is a valuable and much desired asset, the search for failures that lead to its disclosure is an area in constant evolution. Today, communication with all kind of entities is often done through web pages that interact with users, receiving and returning information. These *sites* generally use databases to organize and manage their content. In some cases the dissemination of information is promoted and encouraged, as in the case of marketing, but in others, such for legal or security reasons, its scope should be barred and only accessible to an authorized person. SQL Injections are an effective way to cheat the machine, forcing it to reveal information. This technique uses the dialog boxes (authentication, search ...) of web pages to introduce the attack and induce the machine to reveal, manipulate or even destroy the information.

Information security should not be reduced to a simple computer problem. The alliance of efforts with Information Security in Organizations and the Law are the basis for enhancing more effective security and minimizing the incident consequences, even though reaching an infallible safety goal is hard to achieve.

Training is a factor that really contributes to the correct design and safe use of these systems, either in the theoretical field, in the dissemination of the various concepts, and also in the practical side, through test environments where it is possible to experiment and verify the scope of errors, without real consequences.

Throughout this thesis, a prototype of a security laboratory was developed to demonstrate the cryptographic attacks of Wiener's, Common-modulus and Least Significant Bit, and to test SQL Injection attacks. Servers that wrongly implement ciphers in a virtual machine have been constructed, i.e., which generate the respective parameters neglecting the weaknesses that were detected in the RSA attacks described above, and have also developed the algorithms that make it possible to decipher these randomly generated messages. It was also developed in another virtual machine, a web application that is linked to a database, which is accessed by entering authentication data using the user name and password. This application is vulnerable and allows

to deceive this typical access control, allowing the disclosure of information contained in this database. Finally, a machine identical to the previous one was designed, but in this case, a web application firewall (Mod Security) was mounted, which analyses data entry, detects SQL injections, and blocks such attacks.

These machines were designed for beginners who take the first steps in the area of Information Security, however, allow the future development of additional modules, for testing other type of vulnerabilities.

Keywords: **Cryptography, RSA, Wiener, Common-modulus, LSB, SQLi, Vulnerabilities, Security lab prototype, Tests Environment.**

Índice

Índice de Ilustrações

Símbolos e acrónimos

AES – Advanced Encryption Standard (Sistema criptográfico)

DES – Data Encryption Standard (Sistema criptográfico)

IBM – International Business Machines

Bit – unidade de informação, pode ter um de dois valores 0 ou 1

Byte – Conjunto de 8 bits

CSS - Cascading Style Sheets (estilos de páginas web)

GDPR – General Data Protection Regulation (Regulamento Geral de Protecção de Dados)

HTML - HyperText Markup Language – Linguagem de programação

$\mathbb{N}$ – Conjunto dos números naturais

Directiva NIS – Network and Information Security – Segurança das Redes e da Informação

PHP – Hypertext Preprocessor - Linguagem de programação

SQL – Structured Query Language – Linguagem de programação

SQLi – Structured Query Language injection

U.E. – União Europeia

XOR ou **X⊕R** – Operador lógico que retorna o valor 1 quando os argumentos são diferentes e o valor 0 quando estes são iguais.

XSS injection - Cross-site scripting injection

$\mathbb{Z}$ – Conjunto dos números inteiros

1 Introdução

A evolução tecnológica das últimas décadas facilitou o aparecimento de novas formas de comunicação. A Internet alterou, para um nível planetário, o modo de interacção entre as pessoas, bem como a sua relação com o comércio e os serviços, abrindo o círculo de relações anteriormente limitado pela proximidade física, relacionada com o local onde se vivia, estudava, trabalhava e se realizava qualquer tipo de actividade.

Esta evolução não só obrigou a uma compatibilização organizacional de redes informáticas com arquitecturas distintas, como à necessidade de conciliar sistemas operativos, protocolos e políticas de segurança, entre muitos outros aspectos. Por outro lado, permitiu também colocar grande quantidade de informação no espaço cibernético, facilitando o seu acesso e manipulação, nas mais diversas actividades.

Esta nova forma de interacção digital trouxe enormes vantagens, permitindo agilizar a relação de todo o tipo de comércio e serviços com os seus clientes, nomeadamente optimizando recursos e tempo. Porém, para que todas estas actividades possam ser desenvolvidas é necessário que haja confiança na sua utilização, ou por outras palavras é fundamental que a confidencialidade, a autenticidade e a integridade das mesmas seja assegurada, pois sem esta garantia é inevitável que subsistam dúvidas, legítimas, sobre: “Quem está do outro lado?”; “Estou a falar com quem julgo?”; “É seguro realizar esta operação?”…

Contudo, a par das inúmeras vantagens e muitas vezes estimulado pelo anonimato, ficou também facilitado o aparecimento de formas pouco lícitas de tentar alcançar essa informação.

A garantia de um ciberespaço 100% seguro foi, até à data, impossível de alcançar, e é difícil que venha a existir. Os incidentes podem ter várias origens, desde as falhas técnicas, aos erros involuntários, passando pelos desastres naturais (ANACOM, 2017), mas também podem ser fruto de um acto deliberado com a intenção de provocar dano, de roubar, de extorquir, de fazer uso indevido, de difamar, entre muitas outras. Não existe limite para a imaginação de quem quer explorar estas novas oportunidades. No entanto, mesmo sem garantir uma segurança absoluta, é possível encontrar formas de impedir e amenizar uma parte significativa destas ameaças.

Nos dias de hoje já existe alguma sensibilização, por parte das empresas e entidades, para a adopção de medidas de segurança de caracter técnico, mas é sempre bom ter em atenção que o elo mais fraco e que desencadeia o maior número de falhas é o factor humano, sendo que definir regras claras de utilização, sensibilizar, formar e treinar, são acções que contribuem muito para melhorar a segurança.

Porém, não sendo possível evitar todos os incidentes, ainda assim é possível traçar uma nova linha de defesa, de forma a limitar uma parte dos danos provocados pelos mesmos, utilizando redundância de recursos, como alimentação energética e redes de comunicação alternativas, ou *backups*, entre muitos outros. Todos estes meios devidamente coordenados por um plano de segurança, no qual é pensada e definida, criteriosa e atempadamente, a melhor forma de actuação perante cada cenário, podem ajudar a repor e a normalizar os serviços.

Nos últimos meses têm sido noticiados vários ataques à escala mundial (Vírus Wanna Cry, 12 de Maio de 2017), para os quais a solução aplicada por muitas entidades foi “desligar e esperar que passe” (Ordem dos Engenheiros, 2017). Mas será que essa não é a melhor ajuda que pode ser dada ao Hacker, por exemplo, num ataque de negação de serviço (DOS- Denial of service)? É preciso saber o que fazer e não deixar a resposta ao sabor do improviso.

1.1 Os aliados imprescindíveis da segurança da informação

É na área da Segurança da Informação nas Organizações, que muitas vezes são encontrados os recursos complementares à engenharia informática, necessários para evitar e mitigar as consequências das interferências nefastas nos vários sistemas, através das várias vertentes de sinalização das ameaças e vulnerabilidades conhecidas, ou das boas práticas de segurança física, das redes, das máquinas e do *software* dos sistemas informáticos, passando pela gestão de risco, pela antecipação de cenários, mas também pela estratégia e planeamento da resposta mais adequada a cada incidente.

Não menos importante na cibersegurança, é a conformidade legal e normativa, quer a nível nacional, europeu ou mundial, que por um lado procura balizar e contextualizar o uso dos sistemas informáticos em conformidade com as leis nacionais e por outro, perante o facto de o ciberespaço não ser limitado pelas fronteiras geopolíticas procura regular e uniformizar respostas adequadas, ou melhorar a cooperação nomeadamente dos Estados-Membros da União Europeia. Disto é um exemplo o GDPR – General Data Protection Regulation (Regulamento Geral de Protecção de Dados, Regulamento (UE) 2016/679 do Parlamento Europeu e do Conselho, de 27 de Abril de 2016), que tenta garantir e actualizar à luz das novas tecnologias o direito à privacidade dos dados pessoais. Este regulamento vem suceder à Directiva 95/46/CE, de 24 de Outubro e à Lei nº 67/98, de 26 de Outubro, que a enquadrou na realidade portuguesa, bem como as correspondentes leis nacionais dos restantes Estados-Membros da U.E.

Este regulamento, mais do que aplicar elevadas coimas, aspecto que muito tem preocupado todos aqueles que fazem algum tipo de tratamento de dados pessoais, vem responsabilizar as várias entidades que os tratam, obrigando-as a prestar contas, impondo que a recolha de dados seja feita com o consentimento expresso e informado da finalidade a que se destinam. Vem também designar a nova figura do encarregado de protecção de dados e definir as suas responsabilidades, entre muitos outros aspectos. Vem igualmente consagrar novos direitos, nomeadamente o direito ao esquecimento, aspecto que a meu ver é especialmente relevante no caso particular das crianças.

Um outro exemplo relevante é a Directiva NIS (Network and Information Security Directive, EU 2016/1148 do Parlamento Europeu e do Conselho, de 6 de Julho de 2016), que reflecte a preocupação relativa à segurança das redes e sistemas de informação e que visa melhorar a capacidade e a cooperação na área da cibersegurança entre os Estados-membros da União

Europeia, com especial enfoque na garantia da capacitação mínima dos operadores de serviços essenciais dos sectores da energia, do fornecimento de petróleo, dos transportes, da banca, da saúde e da produção, tratamento e fornecimento de água, bem como na capacitação mínima dos prestadores de serviços digitais importantes, como os motores de busca e computação em nuvem para que tomem medidas adequadas para responderem a incidentes e que reportem os mesmos às autoridades nacionais. Mas também, na designação de um ponto de contacto único nacional, responsável pela coordenação das questões relativas à segurança das redes e dos sistemas de informação e pela cooperação transfronteiriça a nível da União Europeia. (ANACOM, 2017).

A formação em todas estas vertentes é uma peça-chave para alcançar uma segurança mais efectiva, ensinado a prevenir, a actuar perante uma ameaça ou um ataque e a repor a normalidade dos serviços quando as restantes fases falham.

1.2 A formação

A capacitação dos utilizadores através da formação é fundamental, tanto para que possam tirar o melhor partido das ferramentas disponíveis, mas também para que o possam fazer de forma segura, de acordo com as políticas inerentes à organização em que estão inseridos e respeitando as normas legais que regulam a utilização e tratamento dos dados em causa. A formação deve ser adaptada ao tipo de utilizadores, às suas competências e ao seu nível de conhecimento, bem como às especificidades das tarefas a desenvolver na organização. Saber utilizar os sistemas não chega, é necessário consciencializar todos os intervenientes para as questões de segurança, alertando para as potenciais situações a que podem estar sujeitos, incentivando-os a manterem-se informados e actualizados. A aprendizagem em ambientes de teste permite explorar situações de vulnerabilidade, errar e verificar o alcance dessas falhas, voltar a tentar, mas sem consequências reais. Para além disso, permite a todos desenvolverem o espírito crítico sobre o seu próprio sistema, olhando para ele sobre a prespectiva de um atacante que tenta explorar as suas vulnerabilidades.

Existem disponíveis vários ambientes de testes como o WebGoat (OWASP), ou o Natas (Nessos), entre outros, vocacionados para a aprendizagem de vulnerabilidades web, mas que frequentemente são pensados para utilizadores com alguma experiência, sendo pouco acessíveis para utilizadores principiantes que querem aprender de forma autónoma.

Os laboratórios desenvolvidos, no âmbito desta tese, foram pensados para utilizadores que dão os primeiros passos na área da Segurança da Informação, começando pelos fundamentos teóricos básicos sobre criptografia e SQL injection, que servem de apoio para a correcta compreensão dos ambientes de teste apresentados. A forma como estes ambientes foram desenhados permite, futuramente, desenvolver novos módulos para explorar outras vulnerabilidades.

2 A Criptografia

2.1 O uso da criptografia

No mundo actual lidamos com uma enorme quantidade de informação, que pelas mais diversas razões, nem sempre queremos que seja visível por todos ou cuja credibilidade precisamos de garantir. Por essa razão utilizamos a criptografia para poder assegurar a confidencialidade da informação, manter a integridade e a autenticidade dos dados transmitidos, bem como a certificação e autenticação de pessoas e entidades, ou para garantir o não repúdio.

A criptografia é a ciência de escrever secretamente, com o objectivo de esconder o significado de uma mensagem. (Paar & Pelzl, Understanding Cryptography, 2010). A criptografia permite que a existência da mensagem seja conhecida sem que o seu significado seja revelado. Esta técnica difere da esteganografia que camufla a mensagem à vista de todos, para que não seja conhecida a sua existência.

Um sistema criptográfico é um conjunto de algoritmos (regras/instruções) e de chaves secretas que permitem transformar uma informação, que se pretende secreta, em algo incompreensível, mas que para alguém conhecedor desses algoritmos e chaves é facilmente descodificável.

Há sempre alguém, que pelos mais variados motivos quer, ou precisa de aceder a alguma informação secreta e está disposto a tentar quebrar a cifra utilizada. As formas de o fazer podem ir desde a utilização de força bruta, testando todas as hipóteses possíveis, fazendo uso das propriedades algébricas da matemática, até ao aproveitamento da candura, da despreocupação ou mesmo do desconhecimento do alcance das informações reveladas através da engenharia social, ou até mesmo fazendo ataques mais engenhosos como os *Side-Channel Attacks* (ataques ao *hardware*, que se baseiam na observação do comportamento dos dispositivos e componentes, que inadvertidamente revelam informação, como a energia consumida, o tempo necessário para realizar uma tarefa, a radiação electromagnética, as áreas e os componentes usados, ou mesmo o som produzido, entre outros). Não será de estranhar que num futuro próximo sejam descobertas novas e engenhosas formas de atacar uma implementação criptográfica.

A criptologia ou ciência criptográfica está organizada em duas áreas distintas interligadas entre si: a criptografia e a criptanálise. A criptografia, que pode ser subdividida em três grandes áreas: as cifras simétricas, as cifras assimétricas e os protocolos, procura de uma maneira geral encontrar formas seguras de codificar e descodificar mensagens, garantido a confidencialidade, a integridade, a autenticidade e o não repúdio (impossibilidade de o emissor negar a autoria da mensagem). A criptanálise procura encontrar as fragilidades dos vários sistemas criptográficos, com o objecto de decifrar as mensagens, sem conhecer as chaves secretas.

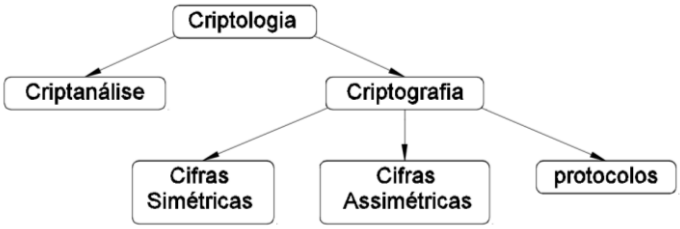

Figura 1 - Componentes da Criptologia (Paar & Pelzl, Understanding Cryptography, 2010)

2.2 A cifragem de mensagens

A necessidade de transmitir uma mensagem secreta, sem que esta possa ser entendida por terceiros, não é uma preocupação recente. Ao longo da história da humanidade, sobretudo por razões militares, foi necessário encontrar formas de transmitir informação, percorrendo caminhos pouco seguros, em que a possibilidade de a comunicação ser interceptada era grande, mas mesmo na hipótese de esta cair em mãos erradas, ainda assim era impossível entendê-la.

De uma forma genérica a cifragem de uma mensagem consiste em:

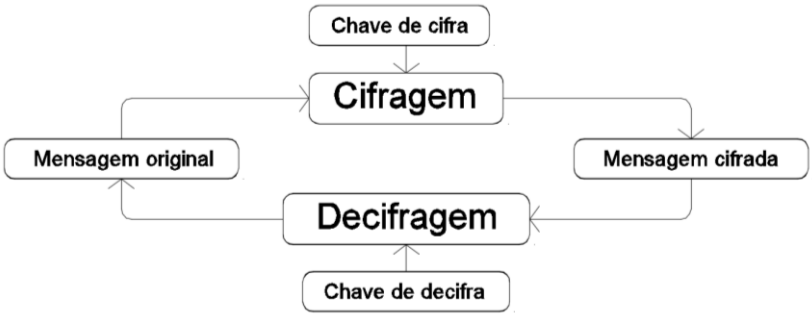

Figura 2 - Esquema de cifragem de uma mensagem

No esquema acima é aplicado um algoritmo que define o mecanismo de transformação da mensagem original, na mensagem secreta e vice-versa, com a particularidade de depender de uma chave ou senha para que essa transformação seja única.

2.3 As Cifras Clássicas (Simétricas)

As primeiras cifras eram realizadas através de mecanismos simples de substituição (confusão), de caracteres por outros, ou de permutação (dispersão) alterando a ordem dos seus elementos, e as chaves secretas eram combinadas previamente, pelos dois interessados, vulgarmente conhecidos nesta área pela Alice (o emissor da mensagem) e pelo Bob (o receptor). A Alice usava o algoritmo e a chave antecipadamente combinados e enviava a mensagem cifrada, pelo canal de comunicação possível na época, e por sua vez o Bob ao receber a

mensagem incompreensível usava a sua chave e o mecanismo que permitiam devolver a mensagem original.

2.3.1 Cifras de Fluxo

Uma das mais célebres cifras de fluxo (*stream cyphers*), que se caracterizam por cifrar um caracter de cada vez, é certamente a Cifra de César, que utilizava a deslocação de três posições do alfabeto i.e., usava a deslocação da posição das letras como algoritmo e uma chave igual a 3, substituindo a letra "A" da mensagem original, pela letra "D" na mensagem cifrada, a letra "B", pela "E" até completar a volta fazendo corresponder a letra "Z" à letra "C".

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Figura 3 - Cifra de César (adaptação para o alfabeto latino actual)

Desta forma a mensagem "BOM DIA" seria cifrada como "ERP GLD", algo incompreensível para quem interceptasse e desconhecesse a lógica da cifragem da mensagem original na informação transmitida. O maior entrave para quem interceptava uma mensagem cifrada desta forma, era certamente o analfabetismo generalizado da época. No entanto, quem tivesse algum conhecimento sobre a lógica da correspondência, mesmo que a deslocação da posição das letras fosse diferente, i.e., que a chave fosse outra, poderia sempre tentar as várias possibilidades e num espaço limitado de tempo poderia decifrar e descobrir a mensagem original.

De uma forma esquemática a cifragem simétrica de uma mensagem pode ser descrita da seguinte forma:

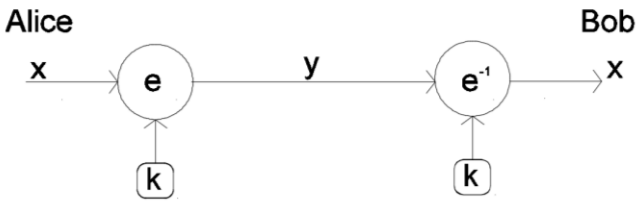

Figura 4 - Princípio da cifragem simétrica ou clássica

A Alice cifra a mensagem usando um algoritmo e uma chave secreta, envia-a para o Bob, que por sua vez decifra a mensagem, usando um algoritmo semelhante ao primeiro, que reverte a mensagem original, usando a chave secreta. Estes sistemas criptográficos pressupõem que o emissor e o receptor da mensagem têm que combinar previamente a chave secreta através de um canal seguro, uma vez que o canal de transmissão pode não o ser (Paar & Pelzl,

Understanding Cryptography, 2010). Por outro lado os pares de chaves de cifra e decifra devem ser únicos entre cada dois utilizadores, pelo que na hipótese de existirem *n* pessoas, são necessárias $n.(n-1)/2$ pares de chaves para possibilitar a comunicação segura entre todos, (Paar & Pelzl, Understanding Cryptography, 2010), elementos esses que devem ser combinados através de um canal seguro e devidamente armazenados.

2.3.2 Cifras de Blocos

Dentro das cifras clássicas ou simétricas, destacam-se também as cifras por blocos (*block cyphers*) que em vez de cifrarem cada caracter sequencialmente, cifram blocos, i.e., conjuntos de caracteres. Estas cifras eram, para a época, substancialmente mais robustas que as *stream cyphers*, mas a evolução da tecnologia e o consequente aumento da velocidade de processamento dos computadores vieram demonstrar, que para algumas destas cifras, a possibilidade de testar todas as hipóteses, i.e., de efectuar um ataque de força bruta era possível em tempo útil. No entanto, para outras como o AES essa capacidade ainda não foi atingida.

2.3.2.1 O DES

Um dos exemplos mais populares destas cifras por blocos, dos últimos trinta anos (Paar & Pelzl, Understanding Cryptography, 2010), é o DES (Data Encryption Standard) que utiliza passos de confusão e passos de difusão, sucessivamente para que a difusão espalhe a confusão o melhor possível.

Esta cifra nasceu da necessidade de encontrar uma forma de standardizar os algoritmos criptográficos nos Estados Unidos da América. Em 1972 foi lançado um concurso, pelo National Bureau of Standard (NBS), actual NIST (National Institute of Standards and Technology), com o objectivo de encontrar um algoritmo seguro que pudesse ser utilizado de forma generalizada (Paar & Pelzl, Understanding Cryptography, 2010). Até esta altura as cifras eram usadas principalmente para fins militares, mas a utilização crescente de computadores fez expandir a necessidade de usar cifras seguras na área empresarial (Zúquete, 2013). O concurso foi ganho por um grupo de criptógrafos que trabalhava para a IBM e o algoritmo baseou-se na cifra de Lucifer, desenvolvida nos finais dos anos 60, por Horst Feiestel.

O DES cifra blocos de 64 bits e usa uma chave de 64 bits. O algoritmo consiste em repetir, dezasseis vezes, um conjunto de operações de confusão, onde cada bit é substituído por outro, e em operações de difusão, onde os bits mudam a posição sequencial em que se encontram. Cada conjunto de repetição é habitualmente designado por *round.* O primeiro passo é dado fazendo uma permutação, i.e., a ordem dos 64 bits do bloco original é baralhada, como se pode ver na figura seguinte:

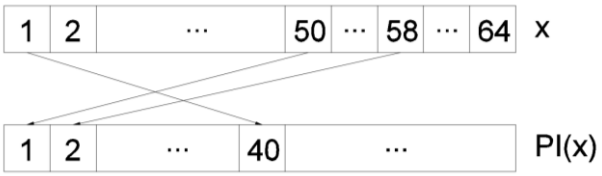

Figura 5 - Permutação inicial do algoritmo do DES (Paar & Pelzl, Understanding Cryptography, 2010)

Esse processo é repetido no final do algoritmo. Na segunda etapa inicia-se o primeiro *round*, o bloco é dividido em duas partes, a esquerda (L_0) e a direita (R_0), e é feita uma troca directa do conjunto de bits do R_0 que é atribuído ao L_1. A metade L_0 é tratada de forma diferente, é transformada por meio de uma função que depende da chave inicial e o resultado desta operação é atribuído à nova metade direita (R_1). De uma forma simples cada *round* pode ser descrito da seguinte forma:

$$\begin{cases} L_i = R_{i-1} \\ R_i = L_{i-1} \oplus f(R_{i-1}, k_i) \end{cases}, com\ i = 1,2,\dots,16 \qquad \text{[eq. 2.3.1] } \textit{(Stinson, 2005)}$$

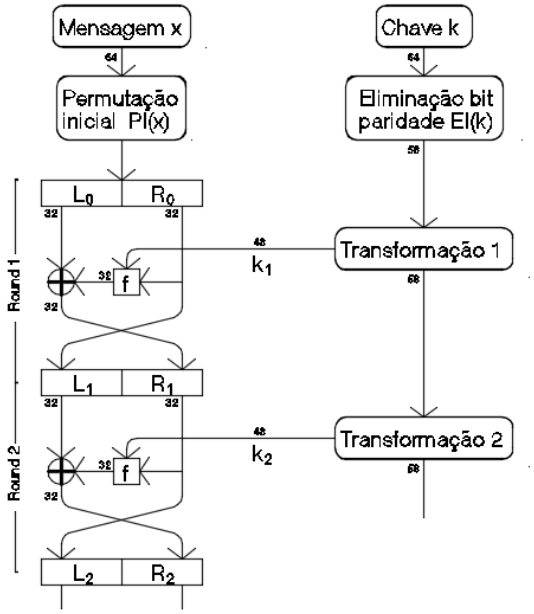

Figura 6 - Primeiros passos do algoritmo do DES (Paar & Pelzl, Understanding Cryptography, 2010)

A chave secreta, por sua vez, também é tratada inicialmente antes de entrar no 1º *round*, sendo eliminados 8 dos seus iniciais 64 bits, mais precisamente o último bit de cada byte (grupo de 8 bits), habitualmente designado por bit de paridade e é realizada a permutação destes 56 bits. Esta chave é posteriormente trabalhada, de forma a gerar as subchaves utilizadas em cada round.

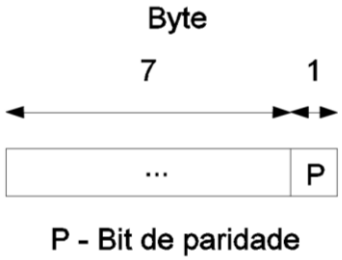

Figura 7 - bits da chave inicial que são eliminados na fase inicial

A função f que, em conjunto com a chave, transforma a metade L_i na parcela R_{i+1}, desempenha o papel principal da segurança deste sistema, tem características de não linearidade ou de imprevisibilidade que dificultam extraordinariamente a sua reversão. A confusão e a difusão dos bits dos vários blocos tornam a criptanálise difícil. Os ataques de força bruta, com todas as suas limitações de eficácia em tempo útil, são no entanto, a solução para quebrar este sistema criptográfico.

2.3.2.2 O AES

Em 1997 foi lançado um novo concurso para desenhar um sistema criptográfico padrão, o AES (Advance Encryption Standard), com o objectivo de substituir o DES por um mais seguro e eficiente. Em 2001, foi escolhido o sistema criptográfico de cifra de blocos Rijndael para blocos de 128 bits, que possibilita o uso de chaves de 128, 192 ou 256 bits, condição exigida no concurso.

Este sistema tem a particularidade de cifrar bytes (conjunto de oito bits), em vez de bits como o DES. O número de *rounds* depende do tamanho da chave, sendo de 10 para uma chave de 128 bits. Cada um destes *rounds* é formado por uma camada de substituição (Byte Substitution Layer) e por uma camada de difusão (Diffusion Layer). A primeira é conseguida através de uma transformação não linear de cada byte (xy), por meio de uma S-Box:

	00	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**0a**	**0b**	**0c**	**0d**	**0e**	**0f**
0	63	7c	77	7b	f2	6b	6f	c5	30	1	67	2b	fe	d7	ab	76
01	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
02	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
03	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
04	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
05	53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
06	d0	ef	aa	fb	43	4d	33	85	45	f9	2	7f	50	3c	9f	a8
07	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
08	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
09	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
0a	e0	32	3a	0a	49	6	24	5c	c2	d3	ac	62	91	95	e4	79
0b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	8
0c	ba	78	**25**	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
0d	70	3e	b5	66	48	3	f6	0e	61	35	57	b9	86	c1	1d	9e
0e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
0f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figura 8- Transformação de cada byte (xy) de acordo com a S-Box

Esta S-Box usa a notação hexadecimal e transforma, por exemplo, o byte C2 no valor 25. A camada de difusão (Diffusion Layer) é feita em dois passos. No primeiro, conhecido por ShiftRow a difusão é feita rodando as linhas de acordo com o esquema abaixo indicado, i.e, considerando que um bloco deste sistema pode ser escrito como uma matriz 4x4, em que cada um dos seus elementos representa um byte. A transformação é conseguida rodando a segunda linha da matriz uma posição para a esquerda, a terceira duas posições para a esquerda e a quarta três para a esquerda.

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

B_0	B_4	B_8	B_{12}	Mantêm
B_5	B_9	B_{13}	B_1	← Deslizam 1 posição para a esquerda
B_{10}	B_{14}	B_2	B_6	← Deslizam 2 posições para a esquerda
B_{15}	B_3	B_7	B_{11}	← Deslizam 3 posições para a esquerda

Figura 9 - Subcamada ShiftRow (Paar & Pelzl, Understanding Cryptography, 2010)

O segundo passo de difusão (MixColumm) consiste numa operação matricial que mistura as colunas da matriz resultante do ShiftRow.

$$\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} X \begin{bmatrix} B_0 \\ B_5 \\ B_1 \\ B_1 \end{bmatrix}$$

Figura 10 - Subcamada MixColumm (Paar & Pelzl, Understanding Cryptography, 2010)

Esta matriz foi cuidadosamente pensada para aumentar a eficiência e poupar tempo na realização desta operação, através da utilização do número neutro da multiplicação, o um.

Desta forma cada *round* do AES pode ser descrito de acordo com o seguinte esquema, em que o bloco a cifrar é formado por 16 bytes (A_0, A_1,…,A_{15}):

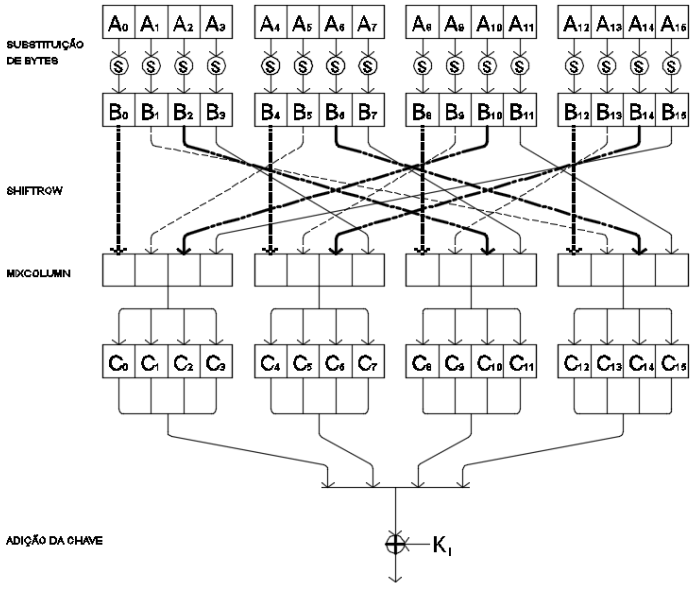

Figura 11 - Round do AES (Paar & Pelzl, Understanding Cryptography, 2010)

As sub-chaves usadas são obtidas a partir da chave anterior e combinadas no final do round através do operador lógico ⊕ (XOR), que retorna o valor 1 quando os argumentos são diferentes e o valor 0 quando estes são iguais.

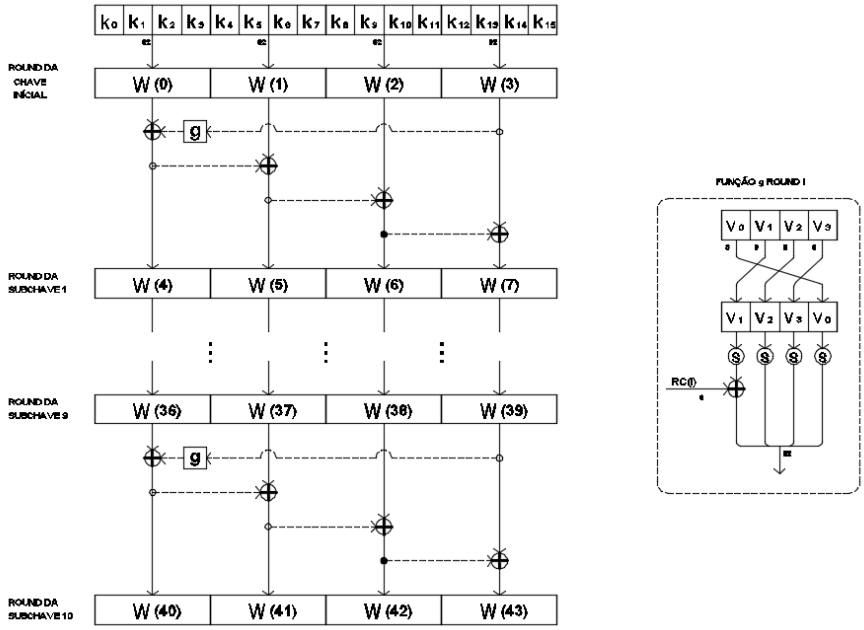

Figura 12 - Esquema de cálculo das sub-chaves de 128 bits (Paar & Pelzl, Understanding Cryptography, 2010)

Tal como nos *rounds* atrás descritos, o esquema de determinação das sub-chaves recorre ao uso de camadas de confusão, utilizando as S-Boxes e ao uso de camadas de difusão.

Esta cifra, que continua a ser utilizada actualmente, é fácil de utilizar quando é conhecida a chave secreta, mas, é extraordinariamente difícil de reverter sem o conhecimento deste elemento, apesar de o seu algoritmo ser publicamente conhecido.

A cifra de César, o DES e o AES são apenas três exemplos de uma multiplicidade de cifras simétricas. As primeiras cifras dependiam do facto de o processo de cifragem ser secreto, (Mao, 2004). No entanto as mais recentes, como o DES e o AES, baseiam-se no Princípio de Kerckhoffs que define que um sistema criptográfico deve ser seguro, mesmo que o atacante saiba todos os detalhes do sistema, i.e. conheça os algoritmos de cifra e decifra, com excepção da chave secreta. (Paar & Pelzl, Understanding Cryptography, 2010) (Fergunson, Schneier, & Kohno, 2010).

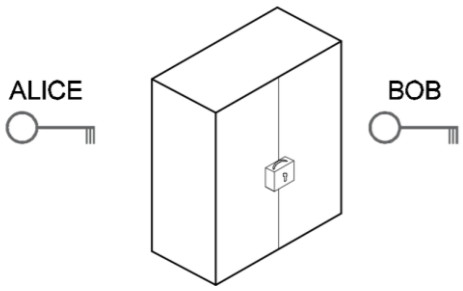

Figura 13 - Analogia para as cifras simétricas. É necessária uma chave para abrir o cadeado (descobrir o segredo)

2.4 As Cifras Assimétricas ou de Chave Pública

A distribuição das chaves, desde sempre atormentou os criptógrafos, existindo ao longo da história muitos exemplos das dificuldades encontradas. Durante a Segunda Guerra Mundial, a complexidade da máquina Enigma trouxe uma vantagem considerável para os alemães, dificultou muito a decifragem das suas mensagens e foi necessário um enorme esforço e investimento por parte dos aliados para descobrirem uma forma de quebrar este sistema tão sofisticado para a época. Um dos pontos fortes deste sistema criptográfico residia no facto de as chaves serem trocadas todos os dias, o que levava a que os esforços para encontrar as regras de transformação das mensagens cifradas nas originais tinha que ser reiniciado todos os dias. Os operadores necessitavam das chaves diárias, que eram distribuídas mensalmente, para poderem operar a máquina Enigma e a hipótese de esta informação cair nas mãos erradas, ou a logística necessária para fazer chegar esta informação crucial aos navios, ou às frentes de combate fragilizavam todo este sistema (Singh, 1999). Foi este obstáculo que impulsionou a procura de formas alternativas de trocar as chaves.

O sistema criptográfico de chave pública foi apresentado em 1976, por Whitfield Diffie e Martin Hellman, e teve o contributo de um trabalho anterior de Ralph Merkle. Baseou-se na ideia de que não é necessário que a chave de cifra seja secreta, e que esta pode ser do conhecimento público, sendo a parte crucial deste processo o facto de o receptor apenas conseguir descodificar a mensagem com a sua chave secreta, que é privada e não é trocada num canal inseguro.

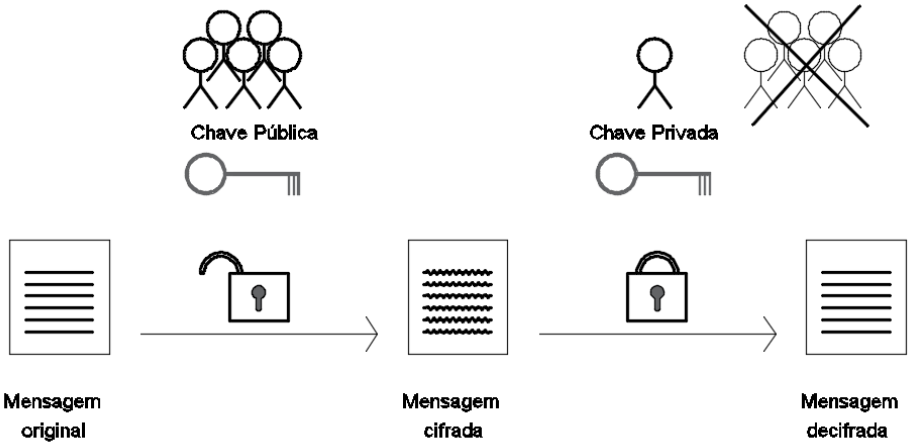

Figura 14 - Sistema de cifragem de chave pública

Este mecanismo descrito para os sistemas criptográficos de chave pública, pode ser usado para várias finalidades, possibilita a geração de chaves secretas em canais de comunicação inseguros, a garantia do não repúdio e da integridade da mensagem, através de algoritmos de assinatura digital, permite também a identificação de entidades, ou pode ser usado para cifrar mensagens utilizando algoritmos como o RSA, que será descrito mais à frente.

2.5 Protocolos

2.5.1 A troca de chaves de Diffie-Hellman

A troca de chaves proposta por Diffie-Hellman veio possibilitar a comunicação sem troca prévia de chaves secretas. A arte deste jogo está em utilizar um algoritmo de cifragem com uma função de sentido único (One –way funtion), i.e., uma função que por um lado é facilmente computável, ou seja, é simples e rápida a cifragem de uma mensagem, mas por outro lado é extremamente complicada de reverter, excepto quando é fornecida uma solução (chave de decifra) tornando-a também simples e célere de verificar, permitindo decifrar a mensagem recebida. São disso exemplo os algoritmos baseados na factorização de números, que são o produto de dois números primos muito grandes, ou no problema de resolver algoritmos discretos num corpo cíclico finito, ou seja na dificuldade de determinar o valor da incógnita **x** em $\alpha^x \equiv \beta mod_p$, conhecendo o valor de **p** e os valores de α (primitiva) e β, ambos pertencentes ao conjunto dos números primos menores que **p-1**, entre outros, como o esquema das Curvas Elípticas (Paar & Pelzl, Understanding Cryptography, 2010). No entanto, é necessário não esquecer que a dinâmica da teoria da complexidade, aliada à descoberta de novas soluções mais eficientes, pode levar a que algumas funções que hoje, embora computáveis, não conseguem responder em tempo útil, passem a poder fazê-lo.

A ideia de base, da proposta de Diffie-Hellman, é que a exponenciação de números primos é uma função de sentido único, por um lado, muito difícil de reverter e por outro é uma operação que goza da propriedade comutativa, i.e., a troca da ordem dos seus argumentos não altera o resultado final. Foi a pensar nestes prossupostos que os dois definiram uma troca segura de chaves.

O processo inicia-se com a escolha de um número primo muito grande p e um outro inteiro α tal que $\alpha \in \{1,2,\dots,p-2\}$, que são do conhecimento público, e o método desenrola-se da seguinte forma:

Alice		Bob
Conhece **p** e **α** Escolhe um número inteiro $\boldsymbol{a} \in \{1,2,\dots,p-2\}$ e calcula $\boldsymbol{A} = \alpha^{a}_{mod_p}$ $\boldsymbol{a} = k_{prv,A}$ é a chave privada da Alice $\boldsymbol{A} = k_{pub,A} \equiv \boldsymbol{\alpha^{a} mod_p}$ a sua chave pública		Conhece **p** e **α** Escolhe um número inteiro $\boldsymbol{b} \in \{1,2,\dots,p-2\}$ e calcula $\boldsymbol{B} = \alpha^{b}_{mod_p}$ $\boldsymbol{b} = k_{prv,B}$ é a chave privada do Bob $\boldsymbol{B} = k_{pub,B} \equiv \boldsymbol{\alpha^{b} mod_p}$ a sua chave pública
	A →	
	← B	
Calcula $k_{ab} = B^{a} = k_{pub}^{\boldsymbol{b}\ \boldsymbol{a}}$		Calcula $k_{ba} = A^{b} = k_{pub}^{\boldsymbol{a}\ \boldsymbol{b}}$

Figura 15 - Troca de chaves de Diffie-Hellman

Na realidade estes dois valores são o mesmo, pois atendendo à propriedade comutativa da exponenciação é possível verificar que:

$$B^{a} = (\alpha^{b}_{mod_p})^{a} = \alpha^{ba}_{mod_p} = \alpha^{ab}_{mod_p} = (\alpha^{a}_{mod_p})^{b} = A^{b}$$

Estes algoritmos têm a desvantagem de ser muitas vezes lentos, pois necessitam de uma capacidade computacional elevada, pouco compatível com a velocidade exigida pelos utilizadores, pelo que frequentemente são utilizados apenas para troca segura das chaves, em alguns casos descartáveis, i.e., de uma só utilização, mas a mensagem é cifrada posteriormente por um sistema criptográfico de chave simétrica, mais rápido de realizar.

3 O sistema criptográfico RSA

3.1 O Algoritmo do RSA

Em 1977, foi proposto por Ronald Rivest, Adi Shamir e Leonard Adleman um novo sistema criptográfico, o RSA, baseado em cifras assimétricas que nos dias de hoje é amplamente utilizado.

A função de sentido único subjacente ao RSA é baseada no problema da factorização dos números inteiros em números primos.

O Algoritmo do sistema criptográfico pode ser sumariamente descrito da seguinte forma (Paar, Lecture 12: The RSA Cryptosystem and Efficient Exponentiation by Christof Paar, 2014):

1 – São escolhidos dois números primos grandes, ***p*** e ***q***.

2 – $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$

3 – $\boldsymbol{\phi}(\mathbf{n}) = (\mathbf{p} - 1).(\mathbf{q} - 1)$

4 – É escolhido um expoente $\mathbf{e} \in \{1,2, \ldots \phi(\mathbf{n}) - 1\}$ de tal forma que o máximo divisor comum $\mathrm{mdc}\big(\mathbf{e}, \boldsymbol{\phi}(\mathbf{n})\big) = 1$

5 – A chave privada é dada por $\mathbf{d} \cdot \mathbf{e} = 1 \mathrm{mod}_{\boldsymbol{\phi}(\mathbf{n})}$

Como resultado ficam definidas as chaves:

A pública $\mathbf{k_{pub}} = (\mathbf{n}, \mathbf{e})$ e a privada $\mathbf{k_{pr}} = (\phi(\mathbf{n}), \mathbf{d})$ (Paar & Pelzl, Understanding Cryptography, 2010),

6 – Com esta informação é possível cifrar a mensagem $\boldsymbol{y} = x^{\boldsymbol{e}} \mathrm{mod}_{\mathbf{n}}$

7 – Com a chave privada é possível descobrir a mensagem original $\boldsymbol{x} = y^{e} \mathrm{mod}_{\mathbf{n}}$

Exemplo prático da aplicação da cifragem com o RSA, utilizando números primos pequenos:

A Alice pretende enviar a mensagem x ao Bob

<u>Alice</u>

Mensagem **x** = 4

<u>Bob</u>

1. Escolhe dois números primos **p** = 3 e **q** = 11

2. Calcula $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$

   $\mathbf{n} = 3 \times 11$

   $\mathbf{n} = 33$

3. Calcula $\phi(\mathbf{n}) = (\mathbf{p} - 1) \times (\mathbf{q} - 1)$

$\phi(\mathbf{n}) = (3 - 1) \times (11 - 1)$

$\phi(\mathbf{n}) = (2) \times (10) = 20$

4. Escolhe um expoente

$\mathbf{e} \in \{1,2, \dots \phi(\mathbf{n}) - 1\}$

$\mathbf{e} \in \{1,2, \dots 19\}$

$\mathbf{e} = 3$

$\mathrm{mdc}(3,20) = 1$

5. A chave privada é então dada por:

$\mathbf{d} \cdot \mathbf{e} = 1\mathrm{mod}_{\phi(\mathbf{n})}$

$\mathbf{d} \cdot \equiv \boldsymbol{e}^{-1} \equiv 7\mathrm{mod}_{20}$

6. O Bob envia então a informação da chave pública à Alice k_{pub} = (**n**,**e**) = (33,3), que usando estes elementos cifra a mensagem da seguinte forma:

$y = x^{\boldsymbol{e}}\mathrm{mod}_{\mathbf{n}}$

$y = 4^3\mathrm{mod}_{33}$

$y = 64\mathrm{mod}_{33}$

$y = 31\mathrm{mod}_{33}$

e envia ao Bob

7. Com a chave de decifra o Bob consegue descobrir a mensagem original

$x = y^{\boldsymbol{d}}\mathrm{mod}_{\mathbf{n}}$

$x = 31^7\mathrm{mod}_{33}$

$x = 27512614111\mathrm{mod}_{33}$

$x = 4\mathrm{mod}_{33}$

O RSA é um algoritmo que requer um processamento elevado de dados e consequentemente é lento, mas pela sua robustez é usado sobretudo para a troca de chaves secretas. Uma vez garantida a confidencialidade das chaves, é frequentemente substituído por outros algoritmos de chave simétrica, que são significativamente mais rápidos. (Paar & Pelzl, Understanding Cryptography, 2010).

O RSA tem outras aplicações para além da troca de chaves, sendo nomeadamente usado em certificados digitais:

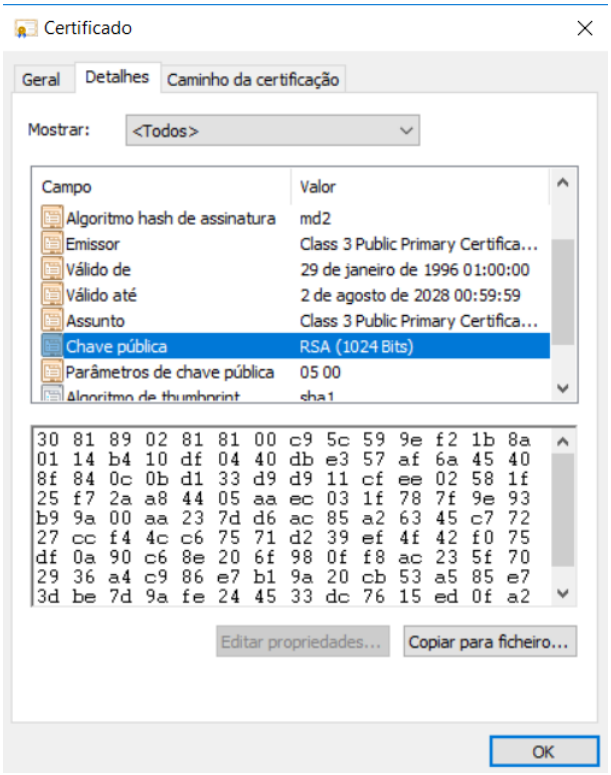

Figura 16 - Certificado Digital – Exemplo de utilização do RSA

Um exemplo típico é aquele que é usado nos *sites* de compras online, que necessitam de uma chave pública única para comunicar com todos os clientes. Este por seu lado precisa de saber se a chave é válida e se é desse mesmo *site*, i.e., que não está a fornecer os dados do seu cartão de crédito a desconhecidos. Esta garantia é dada por uma terceira parte, os CA (Certification Authority). O CA, a pedido do *site* de compras, assina (usando a sua chave privada) um certificado que contém várias informações, nomeadamente a chave pública da comunicação que irá existir entre o futuro cliente e a loja, e envia essa informação para o site de compras. Ao ser solicitada pelo cliente, a loja envia-lhe o certificado que foi fornecido pelo CA. O cliente usa então a chave pública do CA para verificar se a assinatura do certificado é válida. Na prática esta verificação é feita pelo *browser*, que já têm pré-instaladas as chaves públicas dos CA. Agora munido de uma chave pública válida e com a garantia de se tratar do *site* legítimo, o cliente pode enviar os seus elementos, cifrados com a chave pública do site e a loja pode lê-los usando a sua chave privada.

3.2 Criptanálise

Se por um lado a criptografia veio tornar possível enviar mensagens ininteligíveis, por outro a curiosidade e o engenho de quem as intercepta e as pretende descodificar também se foi

aguçando. Nos primórdios das cifras clássicas, atendendo a que os algoritmos utilizados eram simples, era possível analisá-los e quebrá-los, testando todas as hipóteses, ou testando apenas as hipóteses que a análise matemática não permitia eliminar. Com o aumento da complexidade dos algoritmos, mesmo acompanhados pela evolução tecnológica que permitiu um extraordinário incremento da velocidade de verificação de todas as hipóteses, o tempo necessário para quebrar as cifras mais complexas tornou-se por vezes tão grande que inviabiliza que a tarefa seja feita, com sucesso, em tempo útil. Não existem cifras impossíveis de serem quebradas, no entanto algumas são de tal forma árduas de o fazer, requerendo um tempo de tal forma grande que na prática podem ser consideradas seguras.

A criptanálise procura encontrar vulnerabilidades nos algoritmos criptográficos que permitam decifrar as mensagens ou descobrir as chaves secretas. Para tal recorre à análise dos algoritmos e procura encontrar fragilidades que permitam reverter o processo de cifra. Esta pesquisa de informação que pretende devolver a mensagem original, pode também ser realizada de formas mais simples e por vezes mais eficientes, nomeadamente através de engenharia social.

3.3 A Computabilidade e a Complexidade

"O que faz alguns problemas serem dificilmente computáveis e outros não?" (Sipser, 2013). Segundo Sipser esta é a questão central da Teoria da Complexidade e para a qual não existe reposta. As Teorias da Complexidade e da Computabilidade estão relacionadas, a primeira procura classificar os problemas em difíceis ou fáceis e a segunda procura classificá-los de acordo com o facto de terem ou não solução. Na generalidade dos campos, os esforços dos investigadores desenvolvem-se para conseguir chegar a algoritmos simples e rápidos para resolver os mais variados problemas, mas na criptografia esses mesmos esforços vão em sentido contrário, procurando algoritmos complexos, que embora sejam computáveis, requerem um esforço e um tempo tal que na prática inviabilizam a capacidade de encontrar soluções em tempo útil.

A cifra de César, utilizando um alfabeto constituído por 26 letras, necessita apenas de 25 tentativas para testar todas as hipótese possíveis, pelo que é possível testar todas elas usando apenas um lápis e um papel e encontrar a solução num curto espaço de tempo, ou seja é fácil e tem solução. No entanto, uma cifra criptográfica como o RSA, substancialmente mais complexa, que utiliza uma chave secreta com um mínimo de 1024 bits, necessita de verificar 2^{1024} hipóteses, ou seja, embora exista solução e um algoritmo conhecido que descreve como pode ser alcançada, se não for conhecida a chave, esta é muito difícil de descobrir. Apesar da enorme evolução tecnológica que veio permitir o incremento da velocidade de processamento dos computadores, o tempo necessário para testar todas as possibilidades é demasiado grande, tornando inútil tamanho esforço.

A robustez do sistema criptográfico RSA baseia-se no problema da factorização, i.e., na dificuldade de factorizar um número que seja o produto de dois primos muito grandes. Por esta razão, a escolha de **p** e **q** pequenos pode facilitar perigosamente a quebra da cifra.
Se por hipótese **p** e **q** forem pequenos então $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, que é do conhecimento público, é um número que é possível factorizar rapidamente, desta forma $\phi(\mathbf{n}) = (\mathbf{p} - 1).(\mathbf{q} - 1)$ é simples de calcular. Tendo em conta que tal como o **n**, também o valor **e** é público, então é possível calcular a chave privada através $\mathbf{d} \equiv \boldsymbol{e}^{-1} \mathrm{mod}_{\phi(\mathbf{n})}$ e decifrar a mensagem

3.4 Segurança Chave Simétrica vs. Chave Pública

A força de uma cifra está relacionada com a dificuldade de encontrar a sua chave. Este obstáculo depende, por um lado, da complexidade e da eficiência do respectivo algoritmo, por outro depende do comprimento da chave. Para atingir o mesmo nível de segurança, cifras diferentes com algoritmos distintos requerem comprimentos de chaves diferentes.

De acordo com o estudo comparativo apresentado por Mahajan e Sachdeva, sobre o DES, o AES e o RSA é possível observar que o AES tem o algoritmo mais eficiente dos três (Mahajan & Sachdeva, 2013). É aquele que nas condições do estudo é mais rápido a cifrar e a decifrar uma mensagem, razão que nos dias de hoje, certamente está na base da escolha generalizada deste sistema para cifrar mensagens, depois de realizada a troca de chaves. A dificuldade de decifrar uma mensagem, no caso destas duas cifras simétricas, está relacionada com o número de tentativas necessárias para fazer um ataque de força bruta, mas no caso do RSA a questão prende-se com a dificuldade de resolver um problema matemático, menos complexo que o primeiro.

O estudo realizado conclui que nas condições do mesmo, em que o AES apresenta uma chave de 128, 192 ou 256 bits, o DES uma 56 bits e o RSA uma chave com um comprimento mínimo de 1024 bits, o AES é o sistema mais seguro.

A comparação deve ser feita para a generalidade dos casos e não para as situações extremas, em que são conhecidas vulnerabilidades.

Table 1: Comparison between AES, DES and RSA

Factors	AES	DES	RSA
Developed	2000	1977	1978
Key Size	128, 192, 256 bits	56 bits	>1024 bits
Block Size	128 bits	64 bits	Minimum 512 bits
Ciphering & deciphering key	Same	Same	Different
Scalability	Not Scalable	It is scalable algorithm due to varying the key size and Block size.	Not Scalable
Algorithm	Symmetric Algorithm	Symmetric Algorithm	Asymmetric Algorithm
Encryption	Faster	Moderate	Slower
Decryption	Faster	Moderate	Slower
Power Consumption	Low	Low	High
Security	Excellent Secured	Not Secure Enough	Least Secure
Deposit of keys	Needed	Needed	Needed
Inherent Vulnerabilities	Brute Forced Attack	Brute Forced, Linear and differential cryptanalysis attack	Brute Forced and Oracle attack
Key Used	Same key used for Encrypt and Decrypt	Same key used for Encrypt and Decrypt	Different key used for Encrypt and Decrypt
Rounds	10/12/14	16	1
Stimulation Speed	Faster	Faster	Faster
Trojan Horse	Not proved	No	No
Hardware & Software Implementation	Faster	Better in hardware than in software	Not Efficient
Ciphering & Deciphering Algorithm	Different	Different	Same

Figura 17 - Estudo comparativo de segurança: AES, DES e RSA (Mahajan & Sachdeva, 2013)

Segundo Bob Cromwell, citando o estudo comparativo de Schneier e um outro do NIST/NSA, que comparam o comprimento da chave dos vários sistemas para obter o mesmo nível de segurança, é possível observar que uma chave do RSA, com 1024 bits, oferece uma segurança equiparável a uma chave de 80 bits usada num sistema criptográfico de chave simétrica.

Key Length in Bits for Approximately Equal Resistance to Brute-Force Attacks, per Schneier

Symmetric Encryption	56	64	80	112	128	256	448	1024
Cryptographic Hash	112	128	160	224	256	512	896	2048
Asymmetric Encryption	384	494	768	1792	2304	10753	39031	234205

Key Length in Bits for Approximately Equal Resistance to Brute-Force Attacks, per NIST/NSA

Symmetric Encryption	80	112	128	192	256
Elliptic Curve encryption	160	224	256	384	512
RSA (asymmetric encryption)	1024	2048	3072	7680	15380

Figura 18- Comparação do comprimento das chaves para a mesma resistência a ataques de força bruta. (Cromwell, 2017)

4 Ataques ao sistema criptográfico RSA

4.1 Wiener's Attack

Este ataque ao sistema criptográfico RSA, descrito por Michael J. Wiener, mostra uma vulnerabilidade detectada quando a chave privada é pequena (Wiener, 1990), (Stinson, 2005).

A condição necessária para que este ataque seja bem-sucedido é a seguinte:

$$3 \cdot \mathbf{d} < \mathbf{n}^{1/4} \qquad \text{e} \qquad \mathbf{q} < \mathbf{p} < 2 \cdot \mathbf{q}$$

Se **n** tem $\boldsymbol{l}$ bits em representação binária, então o ataque resulta quando **d** tem menos de $\boldsymbol{l}/\mathbf{4} - \mathbf{1}$ bits e em simultâneo **p** e **q** não forem muito afastados.

A opção de escolher uma chave de decifra pequena prende-se com a rapidez que se pode ganhar na decifragem. (Paar & Pelzl, Understanding Cryptography, 2010).

Demonstração:

Condições necessárias para efectuar o ataque

$\boldsymbol{K}_{\mathbf{pub}}$ (**n**,**e**) são públicos; $\qquad 3 \cdot \boldsymbol{d} < \boldsymbol{n}^{1/4} \qquad$ e $\qquad \boldsymbol{q} < \boldsymbol{p} < 2 \cdot \boldsymbol{q}$

Atendendo a que $\mathbf{d} \equiv \boldsymbol{e}^{-1} \mathrm{mod}_{\phi(\mathbf{n})}$ ou escrito de outra forma $\mathbf{d} \cdot \mathbf{e} \equiv 1 \mathrm{mod}_{\phi(\mathbf{n})}$, então:

$\exists_{t \in \mathbb{N}} : \boldsymbol{e} \cdot \boldsymbol{d} - \boldsymbol{t}\boldsymbol{\phi}(\boldsymbol{n}) = 1$

Sendo $\boldsymbol{n} = \boldsymbol{p}.\boldsymbol{q}$, então:

$$q < p < 2 \cdot q$$

$$q \cdot q < p \cdot q < 2 \cdot q \cdot q$$

$$q^2 < n < 2 \cdot q^2$$

$$q < \sqrt{n} < q \cdot \sqrt{2}$$

Consequentemente:

$$0 < n - \phi(n) = p \cdot q - (p-1) \cdot (q-1) =$$

$$= p \cdot q - (p \cdot q - p - q + 1) = p + q - 1$$

$$p + q - 1 < 2 \cdot q + q - 1 < 3 \cdot q < 3\sqrt{n}$$

Ou seja:

$$0 < n - \phi(n) < 3\sqrt{n}$$

Por outro lado:

$$\left|\frac{e}{n} - \frac{t}{d}\right| = \left|\frac{e \cdot d - t \cdot n}{n \cdot d}\right|$$

Tendo em atenção que $e \cdot d = 1 + t\phi(n)$ é possível substituir:

$$\left|\frac{1 + t \cdot \phi(n) - t \cdot n}{n \cdot d}\right| = \left|\frac{1 + t(\phi(n) - n)}{n \cdot d}\right| =$$

$$= \left|\frac{1 - t(n - \phi(n)}{n \cdot d}\right| < \left|\frac{1 - t(3\sqrt{n})}{n \cdot d}\right|$$

$$\left|\frac{1 - 3t\sqrt{n}}{n \cdot d}\right| < \left|\frac{-3t\sqrt{n}}{n \cdot d}\right| = \left|\frac{3t\sqrt{n}}{n \cdot d}\right| = \left|\frac{3t}{\sqrt{n} \cdot d}\right|$$

Como $t < d$ então $3 \cdot t < 3 \cdot d$,

por sua vez, uma das condições de aplicabilidade deste ataque é $3 \cdot d < n^{1/4}$,

logo $3 \cdot t < 3 \cdot d < n^{1/4}$,

pelo que é possível dizer:

$$\left|\frac{e}{n} - \frac{t}{d}\right| < \left|\frac{3t}{\sqrt{n} \cdot d}\right| < \left|\frac{\sqrt[4]{n}}{\sqrt{n} \cdot d}\right| = \left|\frac{1}{\sqrt[4]{n} \cdot d}\right|$$

Por fim é possível concluir:

$$\left|\frac{e}{n} - \frac{t}{d}\right| < \left|\frac{1}{\sqrt[4]{n} \cdot d}\right| < \left|\frac{1}{3 \cdot d \cdot d}\right|$$

$$\left|\frac{e}{n} - \frac{t}{d}\right| < \left|\frac{1}{3 \cdot d^2}\right| < \left|\frac{1}{2 \cdot d^2}\right|$$

Existe uma propriedade da teoria das fracções contínuas, que define que:

se mdc(a,b) = mdc(c,d) = 1 e se

$$\left|\frac{a}{b} - \frac{c}{d}\right| < \left|\frac{1}{2 \cdot d^2}\right|$$

então $\frac{c}{d}$ é uma convergente da expansão da fracção contínua $\frac{a}{b}$.

Sabendo que

$$\left|\frac{e}{n} - \frac{t}{d}\right| < \left|\frac{1}{2 \cdot d^2}\right|$$

é possível calcular $\frac{t}{d}$ como uma convergente da expansão da fracção contínua de $\frac{e}{n}$, atendendo a que **e** e **n** são informação pública.

$$e \cdot d - t\phi(n) = 1$$

$$\frac{e \cdot d}{\phi(n) \cdot d} - \frac{t \cdot \phi(n)}{\phi(n) \cdot d} = \frac{1}{\phi(n) \cdot d}$$

$$\frac{e}{\emptyset(n)} - \frac{t}{d} = \frac{1}{\emptyset(n) \cdot d}$$

Sabendo que **p** e **q** são números grandes, então é possível concluir que:

$$p - 1 \approx p \qquad \text{e} \qquad q - 1 \approx q,$$

pelo que $\phi(n) = (p-1)(q-1) \cong p \cdot q = n$,

por outro lado sendo $\phi(\boldsymbol{n})$ um número grande e **d** um número inteiro, então é legítimo concluir que o seu produto é um número grande, assim sendo:

$$\frac{1}{\emptyset(n) \cdot d} \approx 0$$

Reescrevendo a equação:

$$\frac{e}{\emptyset(n)} - \frac{t}{d} = \frac{1}{\emptyset(n) \cdot d}$$

$$\frac{e}{n} - \frac{t}{d} \approx 0$$

ou seja:

$$\frac{e}{n} \approx \frac{t}{d}$$

Representação de números através de fracções contínuas:

$$q_1 + \frac{1}{q_2 + \frac{1}{q_3 + \cdots + \frac{1}{q_n}}}$$

Desta forma é possível determinar $\phi(\boldsymbol{n})$

Sabendo de antemão que:

$$\phi(n) = \frac{(e \cdot d - 1)}{t}$$

é então possível factorizar o **n**, resolvendo a equação quadrática:

$$\begin{cases} n = p \cdot q \\ \phi(n) = (p-1) \cdot (q-1) \end{cases} \qquad \cdots \qquad \begin{cases} \\ p^2 - [n - \phi(n) + 1] \cdot p + n = 0 \end{cases}$$

Equação esta que pode ser calculada recorrendo à fórmula resolvente:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\text{sendo} \quad \begin{cases} a = 1 \\ b = n - \phi(n) + 1 \\ c = n \end{cases}$$

Desta forma, factorizando **n** nos números primos, **p** e **q**, que o compõem é possível determinar $\phi(\boldsymbol{n})$ e consequentemente calcular a chave privada $\mathbf{d} \equiv \boldsymbol{e}^{-1}\mathrm{mod}_{\phi(\mathbf{n})}$.

Exemplo:

Chave pública K_{pub}(**n**,**e**)=(64741,42667)

$$\phi(n) = \frac{(e \cdot d - 1)}{t} \qquad d = \frac{(t\phi(n) + 1)}{e} \qquad e \cdot d - t\phi(n) = 1$$

$$\frac{e}{n} = \frac{42667}{64741} \approx \frac{t}{d}$$

Cálculo da expansão da fracção contínua $\frac{e}{n}$ usando o algoritmo de Euclides

$$r_{j+1} = r_{j-1} - r_j \cdot q_{j+1}$$

Algoritmo de Euclides

e	n	q=Int(e/n)	r=e-n.q
42667	64741	0	42667
64741	42667	1	22074
42667	22074	1	20593
22074	20593	1	1481
20593	1481	13	1340
1481	1340	1	141
1340	141	9	71
141	71	1	70
71	70	1	1
70	1	70	**0**

t/d	t	d	e	ϕ(n)	
0	0	qualquer ≠ 0	42667	-	-
1	1	1		x	d ≠ 0
1/2	1	2		x	d ≠ par
2/3	2	**3**		64000	d = 3 e ϕ(n) inteiro ok

t/d = $q_1 + \cfrac{1}{q_2 + \cfrac{1}{q_3 + \cdots + \cfrac{1}{q_n}}}$ q = [0, 1, 1, 1, 13, 1, 9, 1, 1,] $\phi(n) = \frac{(e \cdot d - 1)}{t}$

Figura 19 - Algoritmo de Euclides

Admitindo então que **d** = 3 é possível resolver a equação quadrática:

$$\mathrm{x}^2 - [\mathrm{n} - \phi(\mathrm{n}) + 1] \cdot \mathrm{x} + \mathrm{n} = 0$$

e verificar se as suas raízes (**p** e **q**) são números inteiros, tais que $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$

$$\mathrm{x}^2 - [64741 - 64000 + 1] \cdot \mathrm{x} + 64741 = 0$$

$$\mathrm{x}^2 - 742 \cdot \mathrm{x} + 64741 = 0$$

Esta equação tem como raízes $\mathbf{p} = 641$ e $\mathbf{q} = 101$ que são inteiros e verificam $64741 = 641 \cdot 101$

Fica então confirmado que $\boldsymbol{d} = 3$ e a mensagem pode ser decifrada $m = (m^{\boldsymbol{e}})^{\boldsymbol{d}}\mathrm{mod}_{\mathbf{n}}$

4.2 Common Modulus Attack

Se uma mensagem for cifrada e enviada para dois utilizadores cujas chaves públicas são formadas pelo mesmo módulo **n** e as respectivas chaves de cifra são co-primos, então qualquer outro utilizador que tenha acesso às duas mensagens cifradas consegue decifrar o texto (Katz & Lindell, 2008), (Leite, 2009), (Boucinha, 2011)

$\mathrm{n} = \mathrm{p} \cdot \mathrm{q}$; $k_{pub}^{A}(e_A, n)$; $k_{pub}^{B}(e_B, n)$, em que $mdc(e_A, e_B) = 1$

m – mensagem original;

$\begin{cases} c_A = m^{e_A} mod_n \\ c_B = m^{e_B} mod_n \end{cases}$ mensagens cifradas

Então é possível conhecer a mensagem sem o conhecimento prévio das chaves privadas

$k_{pr}^{A}(d_A, \emptyset(n))$, ou $k_{pr}^{B}(d_B, \emptyset(n))$.

Demonstração:

Conhecendo $\boldsymbol{e_A}$ e $\boldsymbol{e_B}$, que são públicos, é possível encontrar dois números inteiros tais que:

$$a \cdot e_{A+} b \cdot e_B = 1,$$

$$C_A^{a_A} \cdot C_B^{a_B} \cong m^{a \cdot e_A} \cdot m^{b \cdot e_B} \cong m^{a \cdot e_A + b \cdot e_B} \cong m^1 \cong m_{mod_n}$$

Usando o Algoritmo de Euclides Estendido (Stinson, 2005) para determinar $\boldsymbol{a_A}$ *e* $\boldsymbol{a_B}$ (este algoritmo é muito usado para calcular o inverso modular)

$$a \cdot e_A + b \cdot e_B = mdc(e_A, e_B) = 1,$$

Exemplo:

$$\mathrm{p} = 11 \text{ e } \mathrm{q} = 43,$$

Mensagem original $\mathrm{m} = 15$

Dados públicos:

$$\mathrm{n} = \mathrm{p} \cdot \mathrm{q} = 473,$$

$$k_{pub}^{A}(e_A, n) = (11; 473),$$

$$k_{pub}^{B}(e_B, n) = (13{,}473),$$

$$mdc(11{,}43) = 1$$

Dados interceptados pelo atacante:

$$\begin{cases} c_A = m^{e_A} mod_n \\ c_B = m^{e_B} mod_n \end{cases} = \begin{cases} c_A = 15^{11} mod_{473} \equiv 246 mod_{473} \\ c_B = 15^{43} mod_{473} \equiv 9 mod_{473} \end{cases}$$

Verificadas as condições necessárias para efectuar o ataque é possível encontrar a mensagem enviada.

$$C_A^{a_A} \cdot C_B^{a_B} \cong m^{a \cdot e_A} \cdot m^{b \cdot e_B} \cong m^{a \cdot e_A + b \cdot e_B} \cong m^1 \cong m_{mod_n}$$

$$a \cdot e_A + b \cdot e_B = mdc(e_A, e_B) = 1,$$

$$a \cdot 11 + b \cdot 43 = 1,$$

Recorrendo ao Algoritmo de Euclides Estendido é possível determinar que:

$$a = 6 \; e \; b = -5$$

$$C_A^{a_A} \cdot C_B^{a_B} \cong 246^6 mod_{473} \cdot 9^{-5} mod_{473} \cong 15_{mod_{473}}$$

4.3 Least-Significant Bit Attack

Este ataque utiliza os dados públicos do sistema criptográfico RSA, $k_{pub}^{A}(\mathbf{e}, \boldsymbol{n})$, para cifrar a mensagem **2**.**m** e pressupõe o prévio conhecimento do valor da mensagem cifrada. Recorrendo a um oráculo de paridade $\boldsymbol{PN_o}$ consegue determinar se o valor calculado é par ou ímpar. Desta forma é possível ir dividindo, ao meio, o intervalo em que se encontra a mensagem e ao fim de várias iterações, consegue determinar qual o bit menos significativo da mensagem original (Mao, 2004).

Aplicando este algoritmo a cada bit da mensagem, i.e., $\log_2 \boldsymbol{n}$ vezes, é possível encontrar a totalidade da mesma.

Um oráculo possível é o utilizado nos Power Analysis Attack of Modular Exponentiation in Smartcards – SEMD – “Single-exponent, Multiple-data” – Thomas S. Messerges (1999) – Cartão que armazena a chave privada e que executa o algoritmo Square & Multiply – este algoritmo é mais lento quando o expoente é impar, porque executa mais operações

Demonstração

$m \in [0, n]$

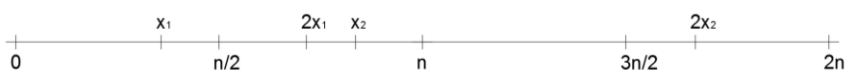

Figura 20- 1º e 2º intervalos de congruência

Se $(2 \cdot m)^e mod_n = 2^e m^e mod_n = 2^e \cdot C mod_n$

for um número par, então **2**.**m** encontra-se no primeiro intervalo de congruência e consequentemente **m** fica situada na primeira metade deste intervalo

$$0 < 2 \cdot m < n$$

$$0 < m < \frac{n}{2}$$

Se $(2 \cdot m)^e mod_n = 2^e m^e mod_n = 2^e \cdot C mod_n$

for um número ímpar então **2**.**m** encontra-se no segundo intervalo de congruência e consequentemente **m** fica situada na segunda metade deste intervalo

$$n < 2 \cdot m < 2 \cdot n$$

$$\frac{n}{2} < m < n$$

Partindo sucessivamente estes intervalos é possível determinar a mensagem **m**.

Dados necessários para fazer o ataque:

$k_{pub}^{A}(\mathrm{e}, n)$;

ter conhecimento da mensagem cifrada $C = m^e mod_n$

existir um oráculo que permita indicar a paridade da mensagem decifrada.

Então:

n é um número inteiro ímpar

$\boldsymbol{m} \in [a, b]$ (CI – intervalo actual)

1ª iteração

$[a, b] = [0, n]$ CI – Intervalo actual

Cifra a mensagem $\mathbf{2} \cdot \boldsymbol{m}$

$(2 \cdot m)^e mod_n = 2^e m^e mod_n = 2^e \cdot C mod_n$

E fornece este valor ao $\boldsymbol{PN_o}$ (Oráculo de paridade)

Se o valor encontrado for par, então a mensagem encontra-se no intervalo $[0, {}^{n}/_{2}]$

Se o valor encontrado for ímpar, então a mensagem encontra-se no intervalo $[{}^{n}/_{2}, n]$

As várias iterações vão-se repetindo, partindo o intervalo ao meio, até que este contenha apenas um número inteiro, sendo essa a mensagem original.

O algoritmo em causa tem sempre as seguintes características:

- Cada iteração reduz o intervalo de possíveis soluções
- A mensagem **m** pertence ao intervalo CI (intervalo actual)

Exemplo:

$k_{pub}^{A}(\mathrm{e}, n) = (3{,}15)$;

$C = m^e mod_n = 13$

Então:

15 é um número inteiro ímpar

$m \in [0{,}15]$

1ª iteração

$[a, b] = [0{,}15]$

Cifra a mensagem $2 \cdot m$

$(2 \cdot m)^e mod_n = 2^e m^e mod_n = 2^e \cdot C mod_n =$

$2^3 \cdot 13 mod_{15} \equiv 14 mod_{15}$

14 é par então:

$a = 0 \ \wedge \ b = n - {}^{n}/_{2} = {}^{15}/_{2}$

$m \in \left[0, {}^{15}/_{2}\right]$ ou seja: $m \in \{1{,}2{,}3{,}4{,}5{,}6{,}7\}$

2ª iteração

$[a, b] = \left[0, {}^{15}/_{2}\right]$

Cifra a mensagem $2 \cdot m$

$(2 \cdot (2 \cdot m))^e mod_n = 2^{2e} m^e mod_n = 2^6 \cdot 13 mod_{15} \equiv 7 mod_{15}$

7 é ímpar então:

$a = 0 + {}^{n}/_{4} = {}^{15}/_{4} \ \wedge \ b = {}^{15}/_{2}$

$m \in \left[{}^{15}/_{4}, {}^{15}/_{2}\right]$ ou seja: $m \in \{4{,}5{,}6{,}7\}$

3ª iteração

$[a, b] = \left[{}^{15}/_{4}, {}^{15}/_{2}\right]$

Cifra a mensagem $2 \cdot m$

$(2 \cdot (2 \cdot (2 \cdot m)))^e mod_n = 2^{3e} m^e mod_n = 2^9 \cdot 13 mod_{15} \equiv 11 mod_{15}$

11 é ímpar então:

$a = {}^{15}/_{4} + {}^{15}/_{8} = {}^{45}/_{8} \ \wedge \ b = {}^{15}/_{2}$

$m \in \left[{}^{45}/_{8}, {}^{15}/_{2}\right]$ ou seja: $m \in \{6{,}7\}$

4ª iteração

$[a, b] = \left[{}^{45}/_{8}, {}^{15}/_{2}\right]$

Cifra a mensagem $2 \cdot m$

$(2 \cdot (2 \cdot (2 \cdot (2 \cdot m))))^e mod_n = 2^{4e} m^e mod_n = 2^{12} \cdot 13 mod_{15} \equiv 13 mod_{15}$

13 é ímpar então:

$$a = {}^{45}/_{8} + {}^{15}/_{16} = {}^{105}/_{16} \wedge b = {}^{15}/_{2}$$

$m \in \left[{}^{105}/_{16}, {}^{15}/_{2}\right]$ ou seja: $m \in \{7\}$

Verificação:

$$C = m^e mod_n \equiv 7^3 mod_{15} \equiv 13 mod_{15}$$

4.4 Outros ataques ao RSA

Os casos anteriormente descritos são apenas uma pequena parte dos ataques conhecidos, que de uma forma geral podem ser classificados como:

1 – Ataques elementares que exploram erros flagrantes no uso indevido do RSA, como o Common Modulus, o Blindig Attack ou a escolha de parâmentro **p** e **q** pequenos;

2 – Ataques que exploram o uso de uma chave privada pequena, de que é exemplo o Wiener. Estes ataques são mais eficazes do que aqueles que é possível fazer perante uma chave pública pequena;

3 – Ataques que exploram o uso de uma chave pública pequena, como o Hastad's Broadcast Attack, o Franklin-Reiter Related Message Attack, ou o Coppersmith's Short Pad Attack. (Boneh, 1999);

4 – Ataques à implementação, de que são exemplo os Timing Attacks. Estes não são dirigidos à estrutura do sistema criptográfico, mas sim à sua implementação. Um desses exemplos é o Power Analysis Attacks of Modular Exponentiation in Smartcards, que através da medição do tempo que um smartcard leva a correr o algoritmo Square and Multiply, com a sua chave secreta, consegue descobrir a chave. Este algoritmo tem a particularidade de ser mais lento a correr um número ímpar (executa mais uma operação), do que um par. (Messerges, Dabbish, & Sloan, 1999). O Least Significant Bit Attack é um exemplo de um ataque à implementação criptográfica.

$$x^n = \begin{cases} x.(x^2)^{\frac{n-1}{2}}, se\ n\ for\ ímpar \\ (x^2)^{\frac{n-1}{2}}, se\ n\ for\ par \end{cases}$$

Figura 21 - Algoritmo Square and Multiply

4.5 Prevenção e contramedidas dos ataques ao RSA

O RSA é o sistema criptográfico *standard* para a cifragem de chave pública e assinaturas e até à data não foi descoberta nenhuma forma eficaz de quebrar este sistema. No entanto, em circunstâncias particulares, como anteriormente foi descrito, é possível quebrar este robusto

sistema criptográfico com base numa má escolha dos seus parâmetros. (Salah, Abdullah, & Oqeilei, 2006).

De acordo com os ataques já mencionados, a escolha dos parâmetros deve ter em atenção os seguintes aspectos:

- O módulo **n**, resultante do produto de dois números primos, deve ser grande, i.e., deve ter pelo menos 1024 bits e não deve ser usado como parte da chave pública de vários utilizadores em simultâneo. Dentro de poucos anos, é natural que a capacidade computacional alcançada venha a obrigar a um aumento deste valor para 2048 bits.

- A chave privada não deve ser pequena. Sendo $\boldsymbol{l}$ o número de bits de **n**, o **d** deve ter pelo menos $\boldsymbol{l}/4 - 1$ bits, i.e., $1024/4 - 1$=255 bits,

- Quando a relação entre **p** e **q** é: $\boldsymbol{q} < \boldsymbol{p} < 2 \cdot \boldsymbol{q}$, a chave privada deve ser tal que: $3 \cdot \boldsymbol{d} > \boldsymbol{n}^{1/4}$.

- No caso dos ataques à implementação do RSA, a escolha criteriosa dos parâmetros não é suficiente para os evitar. No entanto, em casos particulares, é possível adoptar algumas medidas que impossibilitam a determinação da paridade. Um desses exemplos, que pode ser adoptado quando o algoritmo em causa é o Square and Multiply, passa por obrigar o tempo de resposta a ser sempre igual, independentemente da operação que realiza, de forma a inviabilizar a distinção dos bits pares e ímpares. Este algoritmo é mais rápido a processar um bit par porque realiza menos operações.

Existem muitos outros ataques que aproveitam diversas fragilidades dos parâmetros escolhidos, nomeadamente a escolha de uma chave pública pequena, que permitem quebrar o RSA. No entanto, até agora, não foi possível encontrar um algoritmo que quebre este sistema criptográfico de forma eficaz, em tempo útil para todas as situações.

4.6 Protótipo de um laboratório de segurança - Criptografia

Como parte integrante desta tese, foi desenvolvido um protótipo de um laboratório de segurança, baseado nos três ataques anteriormente descritos, que decifra as mensagens que foram interceptadas, nos casos em que as escolhas dos parâmetros foram descuidadas ou quando estes foram usados repetidamente para vários utilizadores. Esta simulação foi realizada em máquinas virtuais e desenvolvidas em Python, com recurso à plataforma de divulgação de código fonte Github (GitHub, 2017).

4.6.1 Roteiro de resolução

Este ambiente de testes foi realizado numa máquina virtual (Oracle's VirtulBox), com o sistema operativo Linux Ubuntu (16.04.2, 64-bit) e com instalação dos módulos Python e Sage.

Para realizar estes testes é necessário instalar a máquina ***Cripto-Attacks*** e executar os comandos indicados para cada um dos ataques descritos de seguida, na directoria respectiva.

4.6.1.1 Wiener's Attack

Objectivo: Demonstrar o ataque de Wiener ao RSA

Descrição: Este ataque foi simulado com recurso a um gerador de números primos aleatórios, que verificam as condições do ataque (***Gerador_Wiener.py***), a um terminal de ataque (***wiener-attack.py***) e a um servidor (***server2_Wiener.py***). Este ataque corre na porta 1234.

Inicializar: Abrir uma janela de terminal (S) e executar na seguinte directoria:

Desktop/CRIPTO/Ubuntu-CRIPTO/Wiener

O comando:

python server2_Wiener.py

que inicia e fica à espera do ataque.

Abrir uma nova janela de terminal (H) e executar na seguinte directoria:

Desktop/CRIPTO/Ubuntu-CRIPTO/Wiener

o comando:

python wiener-attack.py

que inicia o ataque.

Dados: Os dados gerados (**c**,**e**,**n**) são utilizados pelo *hacker* para decifrar a mensagem, de acordo com o algoritmo atrás descrito. Esta mensagem é enviada para o servidor e verificada. Caso esteja correcta é devolvido o seguinte comentário "Correct.".

Resultados esperados:

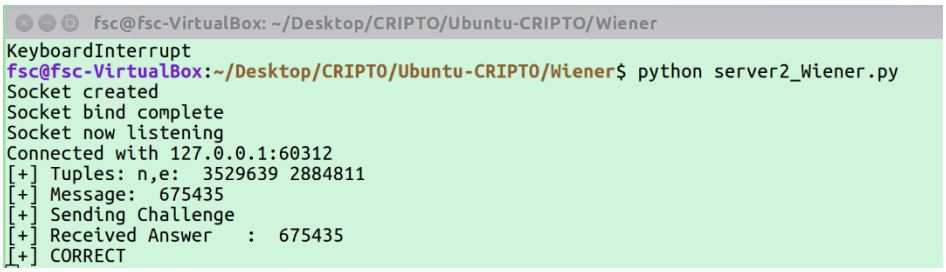

Figura 22 – Wiener's Attack - Terminal (S) do Servidor

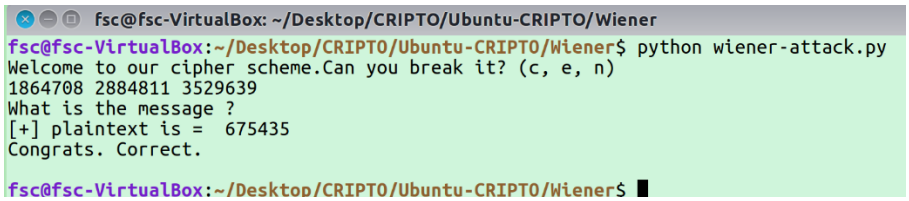

Figura 23 - Wiener's Attack - Terminal (H) do ataque

Erros: Em caso de erro inicializar o servidor (S) e o ataque (H)

4.6.1.2 Common Modulus Attack

Objectivo: Demonstrar o Common Modulus Attack ao RSA.

Descrição: Este ataque foi simulado com recurso a um gerador de números primos aleatórios, que verificam as condições do ataque (***Gerador_Common_Modulus.py***), a um terminal de ataque (***common-modulus-attack.py***) e a um servidor (***server2_CMA.py***). Este ataque corre na porta 1235

Inicializar: Abrir uma janela de terminal (S) e executar na seguinte directoria:

Desktop/CRIPTO/Ubuntu-CRIPTO/Common-Modulus

O comando:

python server2_CMA.py

que inicia e fica à espera do ataque.

Abrir uma nova janela de terminal (H) e executar na seguinte directoria:

Desktop/CRIPTO/Ubuntu-CRIPTO/Common-Modulus

O comando:

sage common-modulus-attack.py

que inicia o ataque

Dados: Os dados gerados (**n**,$\mathbf{c_1}$,$\mathbf{c_2}$,$\mathbf{e_1}$,$\mathbf{e_2}$) são utilizados pelo hacker para decifrar a mensagem, de acordo com o algoritmo atrás descrito. Esta mensagem é enviada para o servidor e verificada. Caso esteja correcta é devolvido o seguinte comentário "CORRECT".

Resultados esperados:

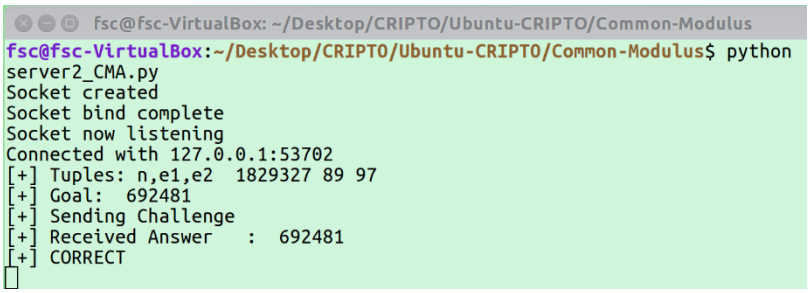

Figura 24- Common-modulus Attack - Terminal (S) do Servidor

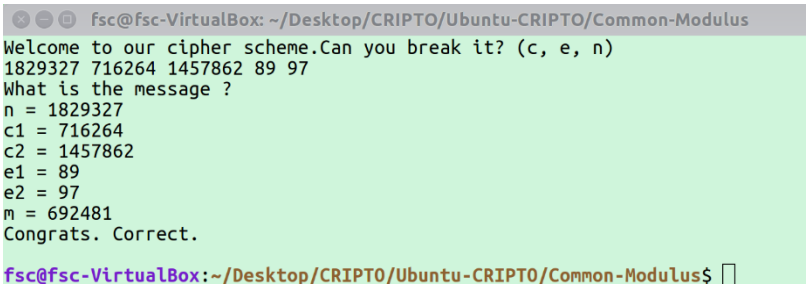

Figura 25 - Common-modulus Attack - Terminal (H) do ataque

Erros: Em caso de erro inicializar o servidor (S) e o ataque (H)

4.6.1.3 Least Significant Bit Attack

Objectivo: Demonstrar o ataque LSB ao RSA

Descrição: Este ataque foi simulado com recurso a um gerador de números primos aleatórios, que verificam as condições do ataque (***Gerador_LSB.py***), a um terminal de ataque (***LSB-Attack.py***) e a um servidor (***server2_LSB.py***). Este ataque corre na porta 1236.

Inicializar: Abrir uma janela de terminal (S) e executar na seguinte directoria:

Desktop/CRIPTO/Ubuntu-CRIPTO/LSB

O comando:

python server2_LSB.py

que inicia e fica à espera do ataque.

Abrir uma nova janela de terminal (H) e executar na seguinte directoria:

Desktop/CRIPTO/Ubuntu-CRIPTO/LSB

O comando:

sage LSB-Attack.py

que inicia o ataque

Dados: Os dados gerados (**c**,**e**,**n**) são utilizados pelo *hacker* para decifrar a mensagem, de acordo com o algoritmo atrás descrito, onde cada iteração é enviada para o servidor que devolve qual a paridade. Este diálogo termina quando o intervalo contém apenas uma mensagem possível e devolve o seguinte comentário "CORRECT".

Resultados esperados:

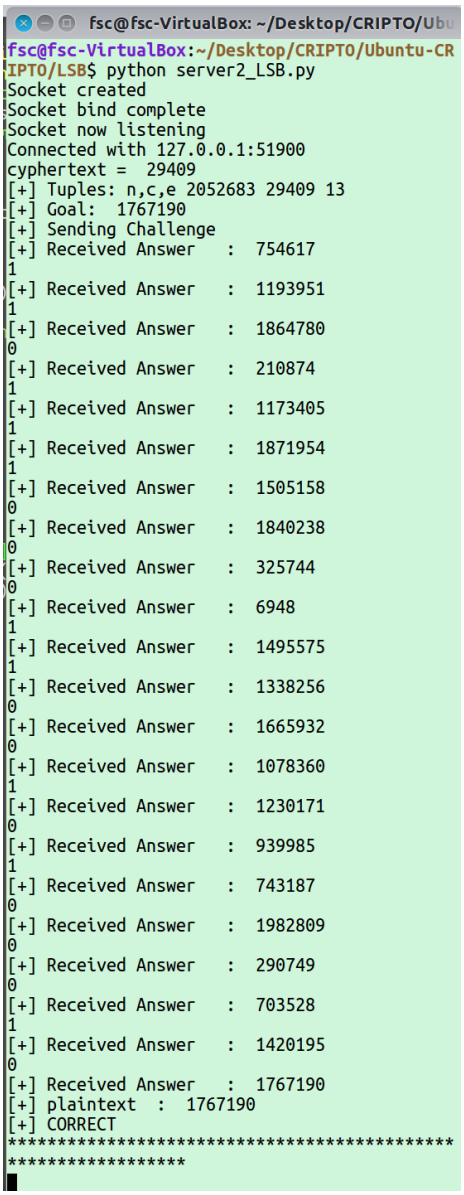

Figura 26 - Least Significant Bit Attack - Terminal (S) do Servidor

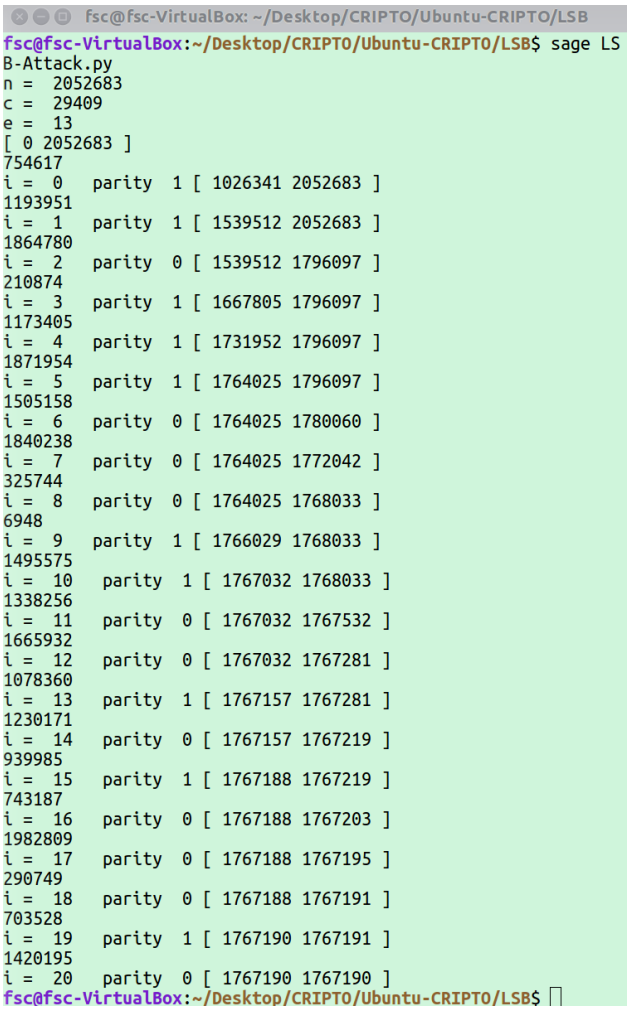

Figura 27 - Least Significant bit Attack - Terminal (H) do ataque

Erros: Em caso de erro reinicializar o servidor (S) e o ataque (H)

Nota: Para não tornar estas demonstrações excessivamente morosas e para que possam ser realizadas num ambiente de formação, limitou-se o valor de $n \in \{números\ primos: 1000 < n < 2000\}$

5 Ataques Web

5.1 Bases de dados

As bases de dados (BD) são uma forma eficiente de guardar, gerir e relacionar informação, podendo ser materializadas por uma simples tabela ou lista, ou mesmo por um conjunto de tabelas relacionadas entre si. Actualmente, o recurso às bases de dados é a forma mais utilizada pelos vários *sites*, para interagirem e trocarem informações com os utilizadores. Estas páginas, em geral, apresentam um interface intuitivo para comunicarem com os clientes/utilizadores, onde é possível fazer pesquisas, ou autenticações para as suas áreas pessoais, i.e., onde é possível fazer algum tipo de introdução de dados. Posteriormente, a página comunica com as respectivas bases de dados e devolve informação, ou valida a permissão de acesso a uma área reservada (Microsoft, 2017).

5.2 Ataques Web

Sendo a informação um bem precioso e muitas vezes imprescindível nas tomadas de decisão de todo e qualquer assunto, é expectável que a vontade de alcançar o seu conhecimento seja tentada de muitas e variadas maneiras, de formas lícitas ou ilícitas. Nas páginas Web em que é possível introduzir algum tipo de dados, estes podem ser usados para ludibriar e manipular os *sites*, levando-os a revelarem e a manusearem a informação, ou a permitirem acessos indevidos, inadvertidamente. O impacto destes ataques depende do valor que é dado à informação guardada, não só do ponto de vista económico, mas também ao nível da reputação pessoal, ou mesmo da reserva da privacidade, para apenas nomear algumas situações.

5.3 O SQL

O SQL (Structured Query Language) é uma linguagem de programação padrão para armazenar, manipular e devolver informação de bases de dados (BD). Com SQL é possível criar bases de dados e respectivas tabelas, introduzir dados, alterá-los e apagá-los, e também fazer pesquisas. As tabelas são organizadas por linhas, a que correspondem os registos e colunas que são designadas por campos. Em geral, as acções sobre as tabelas e BDs são feitas através de instruções que terminam com o sinal ";" (ponto e virgula). Apresenta-se de seguida uma breve lista de instruções que permitem interagir com as bases de dados e que poderão ser usadas no ambiente de testes:

SELECT – Este comando permite fazer uma pesquisa:

```
SELECT * FROM tabela1;
```

Selecciona todos (*) os elementos da tabela1.

INSERT, DELETE e UPDATE são comandos que permitem manipular as tabelas:

```
INSERT INTO tabela1 VALUES (dado do campo 1, …, …);
```

Insere na tabela1 os valores dos dados de cada campo;

```
DELETE FROM tabela1 WHERE id = '1';
```

Apaga o registo identificado por 1, no campo designado por id, da tabela1;

```
UPDATE tabela1 SET NOME = 'Maria' WHERE id = '1';
```

Muda o valor do campo NOME para Maria, do registo identificado como 1 no campo id.

DROP, CREATE são comandos que permitem criar e apagar tabelas:

```
DROP TABLE tabela1;
```

Apaga a tabela1;

```
CREATE TABLE tabela2;
```

Cria a tabela2.

FROM, WHERE ou ORDER BY são cláusulas. Estas condições de selecção são usadas com comandos:

```
SELECT * FROM tabela1;
```

É usada em conjunto com uma selecção ou um comando. Neste caso selecciona todos os registos da tabela1;

```
SELECT * FROM tabela1 WHERE id = '1';
```

Especifica o critério. Neste caso selecciona todos os registos da tabela1 onde o campo id = 1;

```
SELECT * FROM tabela1 ORDER BY nome;
```

Permite ordenar os registos. Neste caso selecciona todos os registos da tabela1 e ordena alfabeticamente pelo campo nome.

Esta breve lista pretende ser meramente ilustrativa do tipo de instruções que a linguagem SQL pode utilizar. Para uma pesquisa mais exaustiva de exemplos é recomendável consultar os *sites* (KhanAcademy, 2017), (W3school, 2017) ou (Codecademy, 2017) entre muitos outros.

5.4 Lista das vulnerabilidades mais frequentes das aplicações Web

De acordo com a OWASP (Open Web Application Security Project), que elabora regularmente a lista das vulnerabilidades mais críticas das aplicações Web (OWASP Top 10

Most Critical Web Application Security Risks) e que conta com contribuições de todo o mundo, as injecções estão no lugar cimeiro deste ranking das vulnerabilidades. Esta lista é corroborada por muitas outras publicações (Sucuri, 2017), (Shah, 2002).

OWASP Top 10 – 2010 (Previous)	OWASP Top 10 – 2013 (New)
A1 – Injection	A1 – Injection
A3 – Broken Authentication and Session Management	A2 – Broken Authentication and Session Management
A2 – Cross-Site Scripting (XSS)	A3 – Cross-Site Scripting (XSS)
A4 – Insecure Direct Object References	A4 – Insecure Direct Object References
A6 – Security Misconfiguration	A5 – Security Misconfiguration
A7 – Insecure Cryptographic Storage – Merged with A9 →	A6 – Sensitive Data Exposure
A8 – Failure to Restrict URL Access – Broadened into →	A7 – Missing Function Level Access Control
A5 – Cross-Site Request Forgery (CSRF)	A8 – Cross-Site Request Forgery (CSRF)
<buried in A6: Security Misconfiguration>	A9 – Using Known Vulnerable Components
A10 – Unvalidated Redirects and Forwards	A10 – Unvalidated Redirects and Forwards
A9 – Insufficient Transport Layer Protection	Merged with 2010-A7 into new 2013-A6

Figura 28 - TOP 10 das vilnerabildades de segurança, em 2010 e 2013, (OWASP, Welcome to QWASP - the free and open software security community, 2017)

5.5 SQL injection (SQLi)

O SQL injection é uma técnica que utiliza a entrada de dados de uma página web, por exemplo o pedido de autenticação de um utilizador, ou o motor de busca de um *site*, para introduzir código, de forma a manipular essa informação. É a forma mais comum de ataque a estas páginas (W3school, 2017). Através destes campos é também possível introduzir malware para controlar outros elementos do sistema, como por exemplo ganhar privilégios de administrador.

5.5.1 Ataques clássicos de SQL Injection

Quando é necessário algum tipo de autorização para aceder a uma área reservada de um site, essa verificação é feita, em geral, através da conferência da correspondência do nome do utilizador e com a respectiva palavra-passe na base de dados do *web site*.

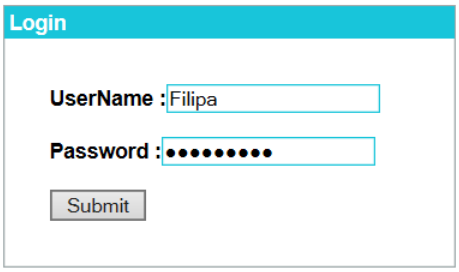

Figura 29 – Exemplo de uma página de autenticação

O SQL injection usa a inserção dos campos da autenticação, para introduzir código que o SQL reconhece como comandos, ou instruções e o levam a revelar ou a alterar a informação da base de dados, inadvertidamente.

Em geral, a verificação da autenticação é feita através de uma condição lógica:

"O USERNAME INTRODUZIDO EXISTE E A PALAVRA-PASSE INTRODUZIDA CORRESPONDE A ESSE USERNAME"

Se o "username" introduzido faz parte da lista existente na base de dados, então esta condição é verdadeira (True) e se a "*password*" digitada corresponde ao utilizador previamente identificado, então esta condição também é verdadeira. A conjunção (e, ∧) é uma operação lógica da matemática que é verdadeira se e somente se as suas condições são verdadeiras.

username	password	username ∧ password
True	True	True
True	False	False
False	True	False
False	False	False

Figura 30 - Tabela de Verdade da Conjunção

A introdução de duas declarações verdadeiras, no username e na *password*, resulta num valor lógico verdadeiro, que é interpretado como: "está reunida a condição necessária para dar acesso, para modificar, ou para revelar informação". Por vezes fá-lo iludido pela lógica da máquina, como se pode ver no exemplo seguinte:

```
uName = getRequestString("username");
```
- recolhe os username digitado

```
uPass = getRequestString("password");
```
- recolhe a *password* digitada

```
sql = "SELECT * FROM tabela_de_utilizadores WHERE Name = 'uName' AND Pass = 'uPass '";
```

- Percorre todas as linhas da "*tabela_de_utilizadores*" e selecciona aquelas cujos valores dos campos Name e Pass correspondem simultaneamente aos valores 'uName' e 'uPass', que foram digitados pelo utilizador.

Os dados introduzidos são atribuídos às variáveis e seguidamente é feita uma pesquisa para seleccionar todos os registos, cujos valores dos campos 'Name' e 'Pass' correspondem simultaneamente aos elementos digitados pelo utilizador. Se for encontrado algum registo que verifique esta condição, então esta condição é verdadeira, ou mais correctamente: a pesquisa é um conjunto não vazio, o que na prática significa por exemplo, que está reunida a condição para validar uma autenticação.

Neste caso verifica a seguinte condição:

```
SELECT * FROM tabela_de_utilizadores WHERE Name =”Filipa” AND Pass =”palavra-passe”
```

Um ataque típico consiste em introduzir um conjunto de caracteres que induza o programa em erro (Error-based SQLi), i.e., que o leve a concluir que a condição que verifica é verdadeira. A arte destes ataques consiste em modificar a condição a verificar, fazendo uso da sintaxe da linguagem SQL

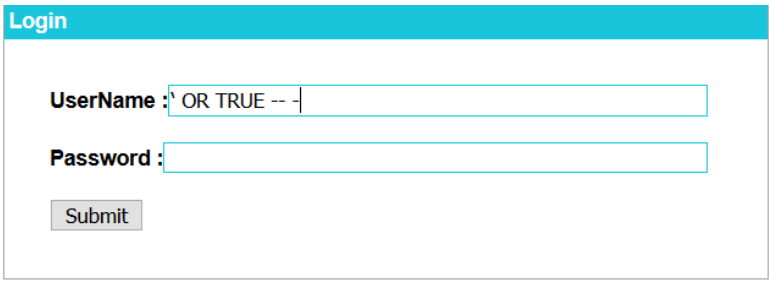

Figura 31 - Ataque clássico de SQL Injection.

```
“SELECT * FROM tabela_de_utilizadores WHERE Name ='' or true -- -' AND Pass ='';
```

A disjunção é uma operação lógica que retorna o valor *false* se, e somente se, as suas condições são simultaneamente falsas. Sendo "true" uma afirmação verdadeira sempre, qualquer que seja a condição que com ela seja conjugada X OR *true*, resulta sempre no valor lógico *true*, pelo que a condição a verificar pode ser resumida da seguinte forma:

(X or true) = true

O SQL utiliza a sequência de caracteres "-- " (dois traços e um espaço) para iniciar um comentário. Desta forma o sentido da *query* original é alterado, pois todas as condições que se encontravam depois destes caracteres especiais são ignoradas.

Assim a máquina lê "`SELECT * FROM tabela_de_utilizadores WHERE Name =''(vazio) or true"` que é sempre uma condição verdadeira para todos os registos. Desta forma conclui que o utilizador reúne os privilégios necessários e autoriza o acesso à informação da base de dados.

Da mesma forma que é possível ludibriar o SQL na autenticação de uma página de uma área reservada, também é exequível a introdução de comandos que podem manipular a base de dados, sendo o seguinte exemplo demonstrativo dessa capacidade:

```
txtUserId = getRequestString("UserId");

TextSQL = "SELECT * FROM Utilizadores WHERE UserId = 'txtUserId'";
```

Tal como na situação anterior são lidos os dados introduzidos e definida a variável que os representa.

User id : 105; DROP TABLE Suppliers

Figura 32 - Inserção de comandos SQL

```
"SELECT * FROM Utilizadores WHERE UserId = '105; DROP TABLE Suppliers'";
```

(W3school, 2017)

Este conjunto de caracteres introduzidos é interpretado como se fossem dois comandos, pois o símbolo ";" é um caracter especial da linguagem SQL que tem a particularidade de finalizar qualquer comando, nomeadamente uma pesquisa. A primeira ordem identifica o utilizador como sendo o "105" e a segunda executa o comando DROP TABLE que descarta, neste caso, a tabela Suppliers. Este é apenas um de muitos comandos que é possível introduzir para manipular e/ou revelar a informação guardada.

Este é o tipo de ataque mais frequente de SQL Injection. O atacante vai alterando a sintaxe da *query* de maneira a conseguir tornar verdadeiras, de forma manipulada, as pesquisas efectuadas pelo SQL, conseguindo autorizações, informações e privilégios vários. A observação e compilação dos resultados desta técnica de tentativa e erro é, frequentemente, o caminho utilizado pelos *hackers* para fazerem um ataque a um site. Os erros indicados podem ter origens diferentes, podem ser fruto de um erro de sintaxe da query, podem resultar da não validação por parte da BD dos valores introduzidos, mas também podem ser resultado da não validação por parte do servidor. Este pode estar preparado para, por exemplo, obrigar a que os campos de inserção de dados sejam de preenchimento obrigatório, ou a forçar que estes sejam constituídos apenas por algarismos, entre muitas outras características (Outsystem, 2017).

Embora este tipo de procedimento não tenha sucesso garantido, pois depende da pesquisa que o *web site* está programado para fazer, existem muitas pistas que são reveladas, de forma indirecta, pelos erros que a verificação produz.

Alguns exemplos de manipulação e da informação que pode ser revelada de uma base de dados (BD1) *construída em Mysql,* que tem uma tabela (tabela1), formada por quatro campos (*id*; *NAME*; *PASSWORD*; email):

id	NAME	PASSWORD	email
1	Vasco	vasco123	vascocg@mail.pt
2	Vera	vera123	veracg@mail.pt
3	Bernardo	bernardo12	bernardocg@mail.pt
4	Tomas	tomas123	tomascg@mail.pt
5	Carmo	carmo12	carmofc@mail.pt
6	Pilar	pilar123	pilarcn@mail.pt
7	Joao	joao123	joaosc@mail.pt
…	…	…	…
17	Francisco	francisco123	franciscofc@mail.pt

Figura 33 - Extracto de uma tabela

Query original para autenticação na área pessoal:

"SELECT ***id*** FROM ***BD1.tabela1*** WHERE ***NAME*** = '$myusername' and ***PASSWORD*** = '$mypassword' ";

Ex 1 - Detecção no número de colunas de uma tabela:

"SELECT ***id*** FROM ***tabela1*** WHERE ***NAME*** =' ' OR true ORDER BY 5 -- -' and ***PASSWORD*** = ' ' ";

Mesmo que não seja visível, o SQL ordena a tabela pela coluna indicada, mas se a coluna não existir então produz um erro. O número de colunas é igual ao valor máximo da coluna a ordenar que não produz erro. A *password* não é verificada pois essa verificação é vista como um comentário.

Ex 2 – Modificar valores a uma tabela:

"SELECT ***id*** FROM ***tabela1*** WHERE ***NAME*** =' ' OR true UPDATE tabela1 SET NAME = 'Vasquinho' WHERE id = '1'; -- -' and ***PASSWORD*** = ' ' ";

Introduz o comando UPDATE, que o SQL reconhece como instrução e actualiza o nome do utilizador identificado na tabela1 como sendo o id=1 para 'Vasquinho'.

Ex 3 – Acrescentar valores a uma tabela:

"SELECT ***id*** FROM ***tabela1*** WHERE ***NAME*** =' ‘ OR true INSERT INTO tabela1 VALUES (NULL, 'Filipa', 'filipa123', ‘filipasc@mail.pt'); -- -‘ and ***PASSWORD*** = ' ' ";

Introduz o comando INSERT INTO, que o SQL reconhece como instrução e insere os dados acima descritos na tabela1.

Ex 4 – Alterar o número de colunas de uma tabela:

"SELECT ***id*** FROM ***tabela1*** WHERE ***NAME*** =' ‘ OR TRUE’; ALTER tabela1 ADD ‘PET’ varchar(30) AFTER NAME; -- -‘ and ***PASSWORD*** = ' ' ";

Introduz o comando ALTER, que o SQL reconhece como instrução e insere uma nova coluna, chamada PET, depois da coluna NAME.

Esta é apenas uma pequena amostra da manipulação que se consegue alcançar com este tipo de ataque. É de referir que o SQL é muito sensível às regras sintácticas, pelo que nem sempre é fácil encontrar a cadeia de caracteres correcta para fazer a manipulação pretendida. A consulta dos *sites* (W3school, 2017), (Codecademy, 2017), (OWASP, Testing for SQL Injection (OTG-INPVAL-005), 2017), (sqlinjection.net, 2017), (pentestmonkey, 2017), (Dougherty, 2012), é útil para identificar exemplos de linguagem SQL e de SQLi que podem ser explorados num ataque a uma página *web*.

5.5.2 Ataques de inferência SQLi

Por vezes o erro resultante da manipulação da query não é visível para o atacante, sendo possível, mesmo assim, fazer algumas deduções reveladoras de informação ou das características da BD. Este tipo de investida denominado *Blind SQL Injection* pode revelar informação útil, quer através do resultado lógico da condição avaliada (Boolean-based), como também através do tempo que demora a realizar uma tarefa (Time-based). No primeiro caso a query pode ser manipulada para devolver um resultado diferente caso a condição seja verdadeira ou falsa. No segundo, a avaliação do tempo que leva a executar uma tarefa pode revelar algumas características da BD.

Um exemplo de aplicação desta técnica pode descrever-se da seguinte forma:

O hacker digita, no campo do utilizador a sequência:

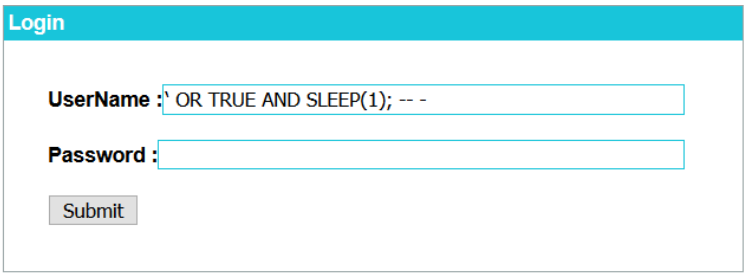

Figura 34 - Ataque de inferência SQL Injection.

Que aplicada à query original, a altera e constrói a seguinte pesquisa:

"SELECT * FROM **family.familia** WHERE **NAME** = ' OR TRUE AND SLEEP(1); -- -"

Neste caso a pesquisa retarda a resposta tantos segundos quanto o número de linhas da tabela.

A função sleep é reconhecida pelo Mysql, programa de gestão de bases de dados que usa a linguagem SQL, e neste caso retarda a sua tarefa, 1 segundo, na verificação de cada registo. O tempo que a pesquisa leva a percorrer todas as linhas da tabela, sem qualquer retardador aparenta ser nulo, pelo que a quantidade de segundos que esta pesquisa retardada leva a ser executada é aproximadamente idêntica ao número de linhas da tabela. Para verificar esta hipótese é possível correr esta mesma pesquisa com tempos diferentes. Saliento, no entanto, que para bases de dados com muitos registos, esta pesquisa pode ser morosa ou imprecisa, quando escolhido um tempo retardador longo ou curto respectivamente. Este ataque pode ser estendido para saber, por exemplo, se o 'administrador' faz parte da tabela, pedindo para retardar 10 segundos se existir um registo com esse nome.

A não validação de *inputs* é uma área vasta que pode ser aplicada a várias linguagens. A detecção e a exploração das vulnerabilidades dependem do sistema utilizado na BD, alvo do ataque, e da programação desenvolvida para a aplicação, mas de uma forma genérica a abordagem de um *hacker* é feita percorrendo os seguintes passos (Stuttard & Pinto, 2011):

1 – Testa a introdução de dados que possam causar má interpretação, ou que provoquem erros na linguagem encontrada;

2 – Identifica e avalia as anomalias das respostas que podem ser indicadores da existência de vulnerabilidades;

3 – Se existirem mensagens explícitas de erro, interpreta-as cuidadosamente;

4 – Introduz variantes dos dados inseridos para aferir a existência da vulnerabilidade que suspeita existir;

5 – Formula a hipótese da existência da vulnerabilidade e prova, testando o maior número de vezes possível, que de facto existe;

6 – Explora as variantes da vulnerabilidade encontrada, para atingir o seu objectivo.

5.6 Contramedidas para evitar as SQL Injections

Tal como já foi mencionado anteriormente, não existem medidas que garantam uma segurança 100% eficaz, mas existem, no entanto, regras de boas práticas para o desenvolvimento deste tipo de sistemas, que minimizam o risco e que podem e devem ser permanentemente actualizadas. Uma forma simples de evitar as situações acima descritas, e que vai na linha das recomendações do lema *Safe by Design and Safe by Default*, passa por impedir a introdução de alguns caracteres especiais, ou por tratar esse *input* como um mero texto, sem qualquer outro significado para além de uma sequência de caracteres, impedindo assim que seja interpretado como um comando. Um exemplo dessa regra é a utilização do comando *mysql_real_escape_string()*, na linguagem PHP, linguagem muito utilizada na comunicação com o servidor, que é um comando eficaz para eliminar o significado especial de certos caracteres, acrescentando o caracter **** (*backslash*) imediatamente antes dos mesmos. Anula, por exemplo, o típico efeito de iniciar um ataque com **‘** (plica), impedindo que seja bem-sucedida a tentativa de alterar o significado original da query.

Nas bases de dados nem sempre é utilizado o Mysql, como nos exemplos acima descritos, mas, utilizando funções que são características apenas de uma das linguagens é possível determinar qual o caminho a seguir, ou melhor, eliminar tentativas inúteis.

Durante o desenvolvimento de uma aplicação a exposição dos erros encontrados é uma ferramenta fundamental para depurar, verificar se o programa procede correctamente e se faz aquilo a que se propõe. Estes comentários devem, no entanto, ser ocultados, ou mesmo apagados nas versões finais, para que não se tornem numa boa fonte de informação, nomeadamente sobre a sua estrutura, para um potencial atacante.

Safe by Design e *Safe by Default* são dois conceitos que podem ser descritos de uma forma simplificada: todo o *software* deve ser pensado e desenvolvido evitando a existência de vulnerabilidades, deve fazer exactamente aquilo a que se propõe e por omissão deve ter a segurança máxima possível. Para além destas regras, os utilizadores devem ter apenas os privilégios de acesso estritamente necessários para desempenharem as suas funções.

5.7 Mod Security

Uma Firewall é um dispositivo que monitoriza e filtra a entrada e saída de tráfego de uma rede, de acordo com regras de segurança pré-definidas. Este filtro, ou escudo, pode ser realizado

através de *hardware*, de *software*, ou ambos. Tal como o nome indica é uma forma eficaz de defender uma rede de intromissões conhecidas e indesejadas (Alecrim, 2017), (Cisco, 2017).

O Mod Security é uma Firewall aplicacional para a web (web application firewall – WAF), em código aberto (open source) desenvolvida pela Trustwave's SpiderLabs (Trustwave Holdings, Mode Security - Open Source Application Firewall, 2017). É uma ferramenta que monitoriza tráfego HTTP, aplicações web, logins (autenticações) e controlos de acesso. Não impõe um padrão de monitorização, dando flexibilidade ao utilizador para definir as regras que pretende adoptar, tanto de detecção como de bloqueio de tráfego suspeito.

Esta aplicação analisa a introdução de dados e verifica se alguma das regras pré-definidas foi quebrada. Quando assim é, regista esses dados no seu ficheiro de logs (modsec_audit.log). Neste arquivo é possível encontrar informação sobre quais as regras transgredidas e numa classificação de 0 (zero) a 10 (dez) o Mod Security indica qual entende ser o ataque: SQLi, XSS, HTTP, etc… (Trustwave Holdings, SpiderLabs/ModSecurity, 2017), (Ristić, 2017), (InfoSecResources, 2017).

É de salientar que esta aplicação está em constante evolução, através da contribuição de utilizadores de todo o mundo, que desta forma ajudam a eliminar os falsos positivos detectados (tráfego que é classificado como transgressor, mas que depois de analisado se constata não o ser) e a definir novas regras.

5.8 Protótipo de um laboratório de segurança – SQL Injection

Como parte integrante desta tese, foi desenvolvido um protótipo de um laboratório de segurança, que utiliza uma página de autenticação de utilizadores e na qual é possível introduzir dados, nomeadamente código malicioso. Este laboratório foi simulado através de duas máquinas virtuais, uma que permite a entrada de dados sem restrições e da qual resulta a possibilidade de autorização indevida numa área reservada, ou na revelação de informação da base de dados e uma segunda que, protegida pela aplicação Mod Security, ferramenta que é compatível com o servidor Apache2, detecta e bloqueia (opção escolhida) a entrada de dados que, segundo as regras pré-estabelecidas, é identificada como código malicioso.

Na construção de uma página web geralmente é necessário um programa para gerir as bases de dados, de que é exemplo o Mysql, uma linguagem para comunicar com o servidor e para tratar a parte gráfica e facilitar o uso são frequentemente utilizas as linguagens HTML e CSS.

Esta simulação foi realizada em máquinas virtuais e desenvolvidas em MySQL (base de dados), PHP (comunicação com o servidor Apache2) e HTML e CSS (página web).

5.8.1 Roteiro de resolução

5.8.1.1 Web_Attacks

Objectivo: Explorar as vulnerabilidades de uma BD, através de SQL Injection.

Requisitos: Máquina virtual (Oracle's VirtualBox) com o sistema operativo Linux Ubuntu (16.04.2, 64-bit), com instalação do servidor Apache2 (*password*: reverse), do Mysql (user: root; *password*: reverse), do PHP e de uma página construída em HTML e CSS (DigitalOcean, 2017), (PHPGroup, 2017).

Descrição: Este ataque foi simulado com recurso a uma base de dados (***family***) que é formada por duas tabelas (***família e guitarras***) e uma página web de autenticação com dois campos: *username* e *password*. O objectivo é testar cadeias de caracteres que provoquem um ataque de SQLi, que iludam a query original, permitindo a autenticação e o acesso fraudulento à página "Welcome", a alteração da BD, ou a revelação de algum tipo de informação.

Inicializar: Instalar a máquina virtual ***Web_Attacks***

Abrir o terminal e executar o seguinte passo para correr o servidor Apache2:

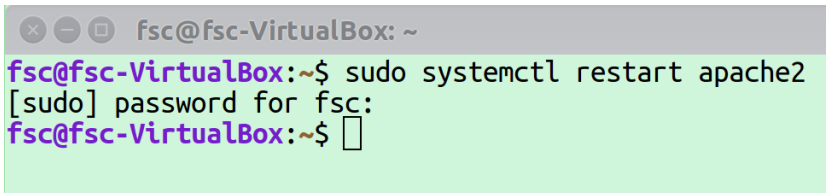

Figura 35 - Iniciar o servidor Apache2

Abrir o Browser Firefox na página:

Figura 36 - Endereço da página web

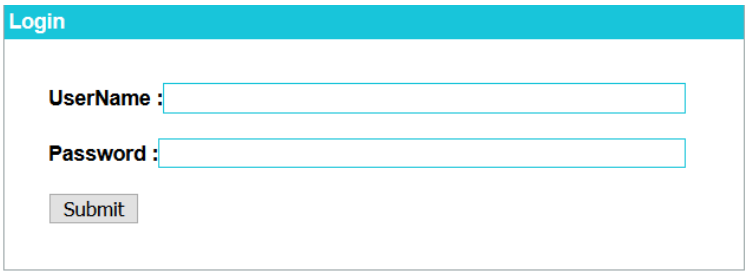

Figura 37 - Página de autenticação do Web_Attacks

Dados: A introdução de dados pode ser feita em ambos os campos. A query existente faz a seguinte pesquisa:

"SELECT * FROM family.familia WHERE NAME = '**$myusername**' and PASSWORD = '**$mypassword**'";

Sugestões de ataques:

1) Username : Vasco
   Password : vasco123

2) Username : ' or true --
   Password :

3) Username : ' OR true ORDER BY NAME; -- -
   Password : 1234

4) Username: ‘ OR TRUE AND SLEEP(1); -- -
   Password :

Resultados esperados:

1) O ‘Vasco’ um é utilizador autorizado a entrar na respectiva área reservada, mediante a apresentação da sua *password*. Nestas áreas reservadas é habitual o seu dono ter acesso aos dados que lhe dizem respeito. A ligação “Sign Out” permite sair da área pessoal e voltar ao menu de Login da página.

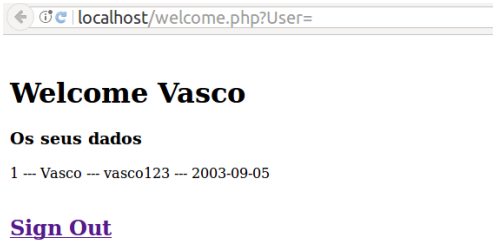

Figura 38 - Resultado de uma autenticação legítima

2) Este ataque revela toda a informação da base de dados e tem a particularidade de assumir que o utilizador é o primeiro nome da lista. Este ponto é particularmente interessante pelo facto de esta posição corresponder frequentemente ao administrador da base de dados, que tem privilégios especiais e pode manipular esta informação com poucas, ou nenhumas restrições.

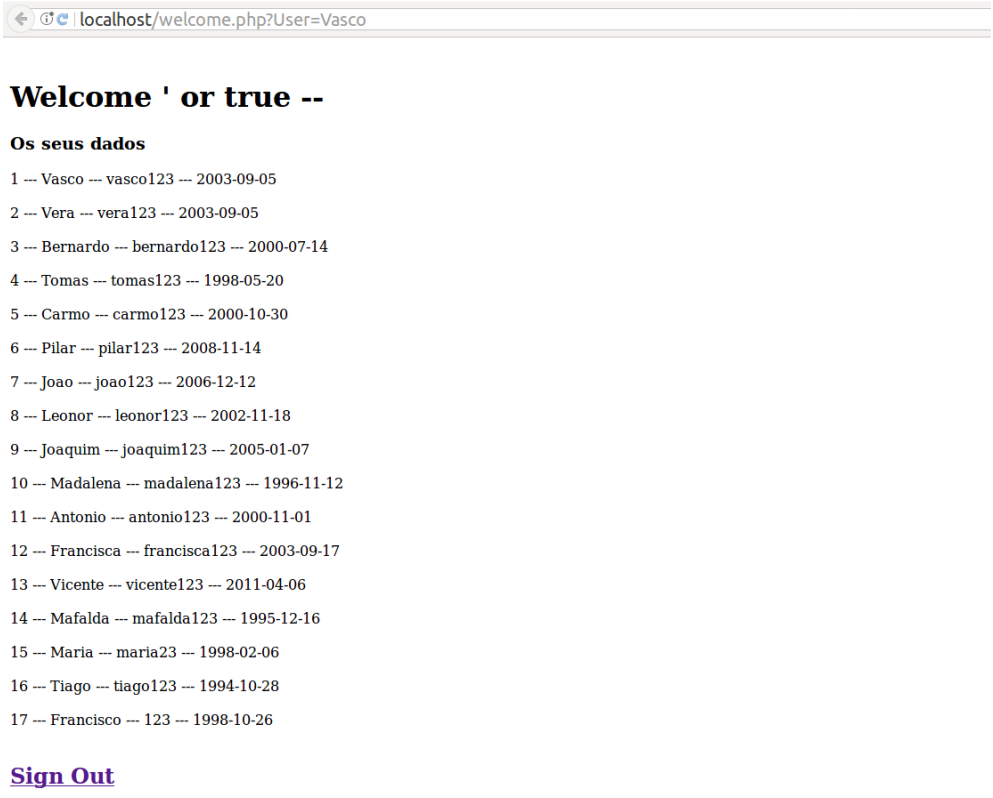

Figura 39 - Resultado do SQLi Username = ‘ or true –

3) Este ataque revela toda a informação da base de dados, ordena-a por ordem alfabética da coluna NAME e assume que o utilizador é o primeiro nome da lista.

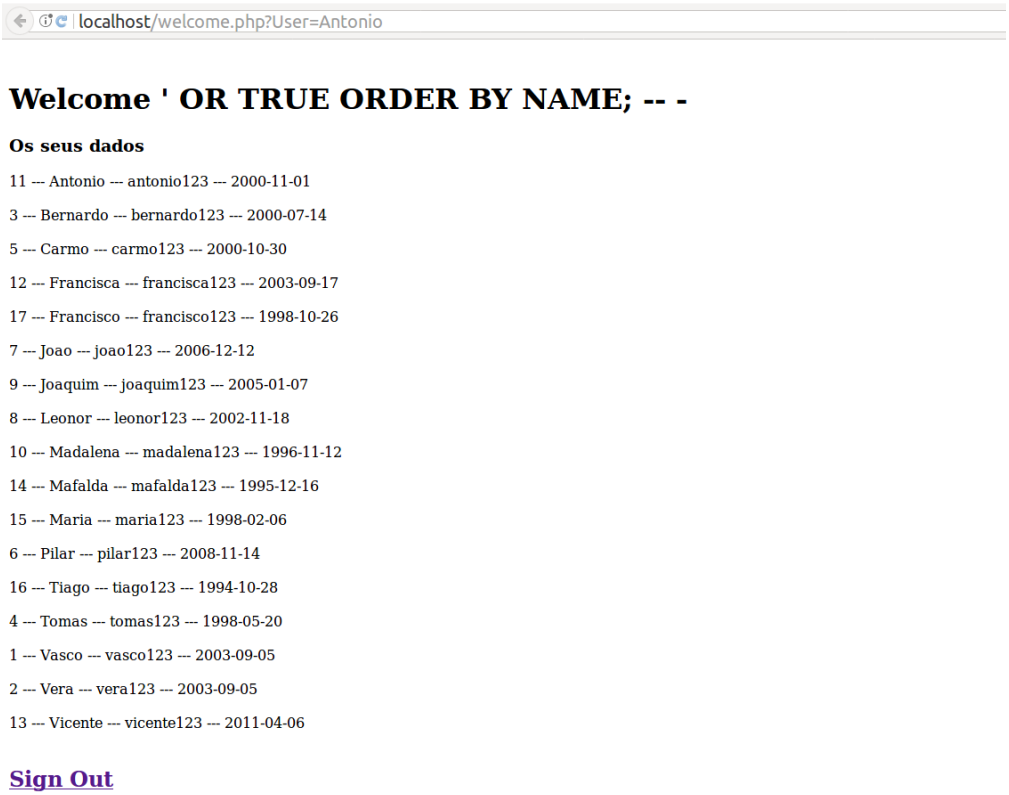

Figura 40- Resultado do SQLi Username = ' OR TRUE ORDER BY NAME; -- -

4) Neste caso não é autorizada a entrada na área reservada.

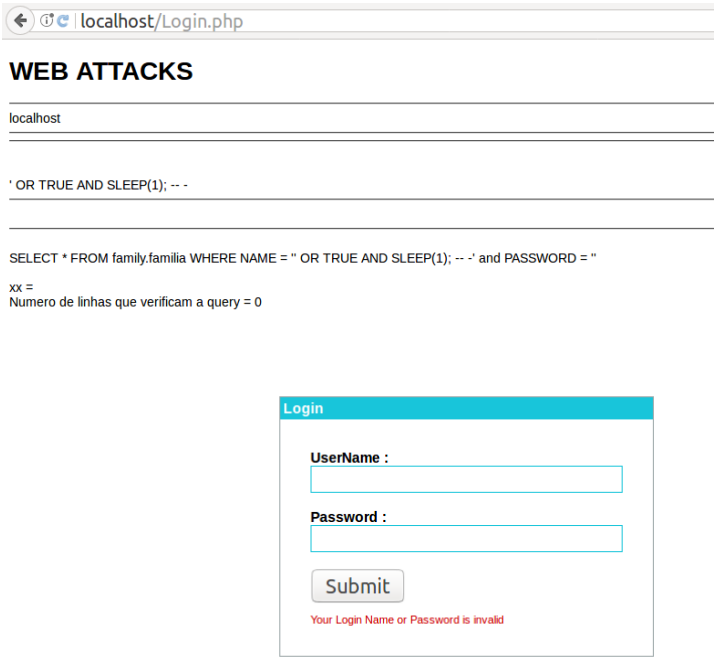

Figura 41 - Resultado do SQLi Username = ' OR TRUE AND SLEEP(1); -- -

Neste último ataque é possível observar que, embora não fiquem reunidas as condições para aceder a uma área reservada, o tempo cronometrado para devolver a resposta de negação de acesso é da ordem dos 18 segundos, indiciando que a tabela deverá ter cerca de 17 ou 18 linhas de informação.

Embora não faça parte do âmbito desta tese, esta aplicação também é vulnerável a outro tipo de ataques como o XSS (Cross-site scripting) Injection, como se pode verificar ao digitar, no formulário, o seguinte código em JavaScript, i.e., comandos na linguagem Java:

http://localhost/?param="><script>alert(“XSS INJECTION !!!”);</script>

(Paola & Carettoni, 2017)

Este conjunto de caracteres é interpretado pelo *browser* como um comando, permitindo de forma indevida, em geral invisível para o utilizador, a execução de acções que não estavam previstas, nem eram pretendidas pelo *site* legítimo. Neste caso é aberta uma caixa com a mensagem “XSS INJECTION!!!”.

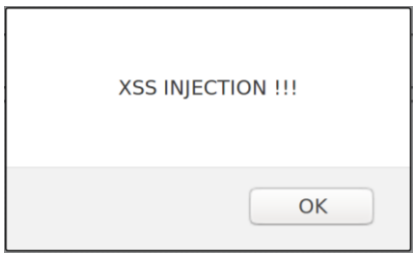

Figura 42 - Exemplo de XSS Injection

5.8.1.2 Web_Attacks+ModSecurity

Objectivo: Explorar as vulnerabilidades de uma BD, através de SQL Injection, na presença de uma aplicação de firewall (Mod Security)

Requisitos: Máquina virtual (Oracle’s VirtualBox) com o sistema operativo Linux Ubuntu (16.04.2, 64-bit), com instalação do servidor Apache2 (*password*: reverse), do Mysql (user: root; *password*: reverse), do PHP, uma página web em HTML e CSS, e da aplicação Mod Security.

Descrição: Este ataque foi simulado com recurso a uma base de dados (***family***) que é formada por duas tabelas (***família e guitarras***) e uma página web de autenticação com dois campos: username e *password*, mas na presença de uma firewall. O objectivo é testar cadeias de caracteres que provoquem um ataque de SQLi. Que iludam a query original, permitindo a autenticação e o

acesso fraudulento à página “Welcome”, a alteração da BD, ou a revelação de algum tipo de informação e observar a reacção da firewall.

Inicializar: Instalar a máquina virtual ***Web_Attacks+ModSecurity***
Abrir o terminal e inicializar o servidor Apache2.
Abrir o Browser Firefox na página: localhost/Login.php

Dados: A introdução de dados pode ser feita em ambos os campos.

Sugestões de ataques:

1) Username : Vasco
Password : vasco123

2) Username : ’ or true --
Password :

3) Username : ' OR true ORDER BY NAME; -- -
Password : 1234

4) Username: ‘ OR TRUE AND SLEEP(1); -- -
Password :

Resultados esperados:

1) O ‘Vasco’ é um utilizador autorizado a entrar na respectiva área reservada, mediante a apresentação da sua *password*. Nestas áreas é habitual o seu dono ter acesso aos dados que lhe dizem respeito. A ligação “Sign Out” permite sair da área reservada e voltar ao menu de Login da página. Neste caso não há qualquer manifestação visível da aplicação Mod Security.

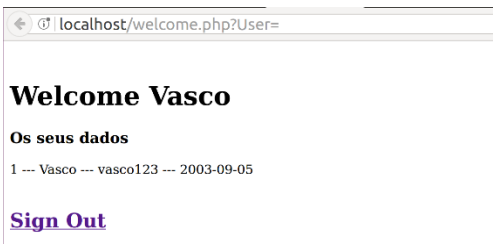

Figura 43 - Resultado de uma autenticação legítima na presença do Mod Security

2) Neste ataque é infringida pelo menos uma das regras pré-definidas. O Mod Security bloqueia a página e regista o evento no ficheiro modsec_audit.log.

Figura 44 - Resultado do SQLi Username = / or true -- na presença do Mod Security

3) Tal como no caso anterior é infringida pelo menos uma das regras pré-definidas. O Mod Security bloqueia a página e mais uma vez regista o evento no ficheiro modsec_audit.log.

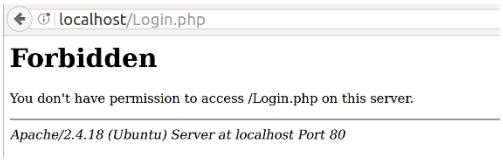

Figura 45- Resultado do SQLi Username = ' OR TRUE ORDER BY NAME; -- - na presença do Mod Security

4) Neste caso, a resposta é imediata, não leva cerca de dezassete ou dezoito segundos como acontece na máquina sem o Mod Security instalado. Este bloqueia a página e mais uma vez regista o evento no ficheiro modsec_audit.log.

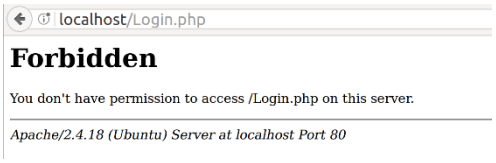

Figura 46 - Resultado do SQLi Username = ' OR TRUE AND SLEEP(1); -- - na presença do Mod Security

Ao analisar o ficheiro modsec_audit.log, que regista a introdução de dados maliciosos, é possível verificar que o Mod Security identifica estas entradas como SQLi, com a classificação máxima (10).

Excerto do **modsec_audit.log em** `resposta ao ataque: ' or true -- -`

```
[11/Oct/2017:07:23:47 +0100] Wd24838AAQEAAE8R4s8AAAAE 127.0.0.1 42440
127.0.0.1 80
--a0b60525-B--
POST /Login.php HTTP/1.1
```

```
Host: localhost

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:51.0) Gecko/20100101 Firefox/51.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: http://localhost/Login.php

Cookie: PHPSESSID=7hfn03gkoeghqbtvu3rpqfhlo4

Connection: keep-alive

Upgrade-Insecure-Requests: 1

Content-Type: application/x-www-form-urlencoded

Content-Length: 31


--a0b60525-C--

NAME=%27+or+true+--+-&PASSWORD=

--a0b60525-F--

HTTP/1.1 403 Forbidden

Content-Length: 293

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: text/html; charset=iso-8859-1


--a0b60525-E--

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html><head>

<title>403 Forbidden</title>

</head><body>

<h1>Forbidden</h1>

<p>You don't have permission to access /Login.php

on this server.<br />

</p>

<hr>

<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>

</body></html>
```

--a0b60525-H--

Message: Warning. detected SQLi using libinjection with fingerprint 's&1c' [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "68"] [id "942100"] [rev "1"] [msg "SQL Injection Attack Detected via libinjection"] [data "Matched Data: s&1c found within ARGS:NAME: ' or true -- -"] [severity "CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity "1"] [accuracy "8"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"]

Message: Warning. detected SQLi using libinjection with fingerprint 's&1' [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "68"] [id "942100"] [rev "1"] [msg "SQL Injection Attack Detected via libinjection"] [data "Matched Data: s&1 found within ARGS:NAME: ' or true "] [severity "CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity "1"] [accuracy "8"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"]

Message: Access denied with code 403 (phase 2). Operator GE matched 5 at TX:anomaly_score. [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "57"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Total Score: 10)"] [severity "CRITICAL"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-generic"]

Message: Warning. Operator GE matched 5 at TX:inbound_anomaly_score. [file "/etc/apache2/owasp-modsecurity-crs/rules/RESPONSE-980-CORRELATION.conf"] [line "73"] [id "980130"] [msg "Inbound Anomaly Score Exceeded (Total Inbound Score: 10 - **SQLI=10**,XSS=0,RFI=0,LFI=0,RCE=0,PHPI=0,HTTP=0,SESS=0): **SQL Injection Attack Detected** via libinjection"] [tag "event-correlation"]

Action: Intercepted (phase 2)

Apache-Handler: application/x-httpd-php

Stopwatch: 1507703027910545 21189 (- - -)

Stopwatch2: 1507703027910545 21189; combined=14426, p1=10421, p2=3939, p3=0, p4=0, p5=66, sr=19, sw=0, l=0, gc=0

Response-Body-Transformed: Dechunked

Producer: ModSecurity for Apache/2.9.0 (http://www.modsecurity.org/); OWASP_CRS/3.0.2.

Server: Apache/2.4.18 (Ubuntu)

Engine-Mode: "ENABLED"

--a0b60525-Z--

6 Conclusão

A evolução tecnológica na área digital revolucionou a forma de comunicação entre as pessoas, a Internet agilizou o seu relacionamento com o mundo, trazendo enormes vantagens na comunicação com os serviços, ou em áreas como o ensino, a saúde, a ligação a entidades oficiais e um sem fim de outras situações. Para que este tipo de sistema funcione e nele se possa confiar, é necessário que a confidencialidade, a autenticidade e a integridade das comunicações seja assegurada. A criptografia é uma arma poderosa para alcançar esses objectivos. Esta ciência antiga, que se adaptou aos tempos modernos, desenvolveu cifras complexas, cujos algoritmos de decifra, embora computáveis, são extremamente difíceis de reverter sem o conhecimento das chaves secretas. Não obstante essa dificuldade, existem situações em que a robustez de uma cifra, e em particular do sistema criptográfico RSA, pode ficar comprometida quando é feita uma selecção de parâmetros pouco criteriosa. Os ataques de Wiener, Common Módulos e o LSB são exemplos de fragilidades que têm origem na escolha de uma chave privada pequena, ou na opção de um módulo **n**, comum a vários utilizadores, cujas chaves públicas são co-primos, ou em vulnerabilidades distintas da escolha dos parâmetros, mas feitas directamente à implementação criptográfica, que fragilizam o RSA e possibilitam a decifra, em tempo útil, das mensagens interceptadas.

A segurança das comunicações pode ser colocada em causa de muitas outras formas. A Informação é um bem valioso, muito desejado e fundamental em muitas tomadas de decisão. As bases de dados são uma forma eficiente de guardar, gerir e relacionar a informação. Os *web sites* e as respectivas páginas recorrem habitualmente às bases de dados para interagirem com os utilizadores, permitindo-lhe fazer autenticações, ou responder a pesquisas, entre outras funcionalidades, embora limitados pelos privilégios de acesso concedidos a cada um, visto que alguns conteúdos não devem estar ao alcance de todos. O relacionamento dos utilizadores com estas páginas é habitualmente feito através da entrada de dados num espaço de pesquisa, ou numa zona de autenticação para áreas de acesso restrito, como as áreas reservadas. Em resposta, os utilizadores obtêm o privilégio de entrar nessas áreas ou de receber algum tipo de informação a qual estão autorizados a ver. Estes campos de introdução de dados podem ser utlizados, quando o seu conteúdo não é devidamente validado, por alguém mal-intencionado, para ludibriar a máquina, obrigando-a a revelar informações. O SQL Injection é uma técnica eficaz de manipular as pesquisas realizadas numa base de dados, de maneira a revelar e manusear informação ou a permitir acessos indevidos, de forma inadvertida. O impacto destes ataques depende do valor que é dado à informação guardada, não só do ponto de vista económico, mas também ao nível da reputação pessoal, ou relativo à privacidade, para nomear apenas algumas situações. A construção de aplicações que cumpram as regras Safe by Default e Safe by Design é um bom princípio. A incorporação, no código, de instruções que impeçam, ou anulem o significado de alguns caracteres especiais, ou que obriguem a que os dados introduzidos sejam interpretados como caracteres, e não como comandos do SQL, são cuidados que devem estar sempre presentes na programação de um *site.* Melhor ainda é aliar essas regras

a uma firewall aplicacional que detecte ou bloqueie um eventual código malicioso, como é o caso da aplicação Mod Security. Esta, que de acordo com as regras escolhidas pode, por exemplo, impedir também a introdução de caracteres com um significado especial em SQL, ou ignorar a introdução de comandos, consegue igualmente alertar para outro tipo de ataques, nomeadamente XSS injection. Esta aplicação pode ajudar os programadores a colmatar as vulnerabilidades que vão surgindo ao longo da vida útil dos programas, aproveitando a contribuição da comunidade, que vai descobrindo novas vulnerabilidades. A aplicação destas medidas ajuda a tornar mais robusta a segurança no acesso às bases de dados e contribui para que a informação não seja revelada inadvertidamente.

A segurança da informação não deve ser reduzida a um simples problema informático. A cooperação de esforços das áreas da segurança organizacional e do Direito são essenciais para complementar a área técnica da segurança informática. O contributo da Segurança de Informação nas Organizações através da sinalização de ameaças e vulnerabilidades conhecidas, da divulgação das boas práticas de segurança física, das redes, das máquinas e do *software* dos sistemas informáticos, passando pela gestão dos riscos associados e pela estratégia e planeamento de respostas adequadas a cada tipo de incidente, são ajudas fundamentais para aumentar o nível de segurança. Não menos importante é a contribuição do Direito, balizando e contextualizando o uso dos sistemas informáticos em conformidade com as leis nacionais, procurando através da harmonização de regras e da cooperação dos Estados-membros da União Europeia, responder ao facto de este espaço não ser limitado pelas fonteiras geopolíticas, ou definindo a capacitação mínima dos operadores de serviços essenciais (Energia, Transportes, banca, saúde e produção, tratamento e abastecimento de água) e dos prestadores de serviços digitais importantes, são efectivamente contribuições que se complementam e ajudam a manter o ciberespaço mais seguro. Por último, mas não menos importante, é o valioso contributo da formação contínua dos utilizadores, que são peças cruciais para potenciar uma segurança mais eficaz e minimizar as consequências dos incidentes, pois sem esta peça chave todas as outras tendem a falhar. O conhecimento é a base para colocar em prática o uso correcto de todos estes sistemas, mesmo sabendo que o objectivo de tornar a segurança infalível é de difícil alcance.

A aprendizagem teórica e prática das regras que contribuem para uma utilização segura dos sistemas informáticos é fundamental. A possibilidade de poder experimentar e verificar o alcance dos erros, em ambientes de teste sem consequências reais, é uma forma eficaz de aprender. Tendo por alvo os utilizadores que dão os primeiros passos na área da Segurança da Informação, e depois de apresentados os fundamentos teóricos que ajudam a compreender os laboratórios propostos, foram desenvolvidos ambientes de testes que simulam as vulnerabilidades do sistema criptográfico RSA (os ataques de Wiener, de Common Modulus e de LSB) e que simulam uma página vulnerável a ataques de SQLi. Nos primeiros são gerados os vários parâmetros, de acordo com as características de cada um, e é possível demonstrar que, na presença das vulnerabilidades descritas, é exequível a decifra das mensagens. No segundo foi construída uma página de autenticação que permite a introdução de dados e onde

é possível testar ataques de SQLi. O protótipo desenvolvido permite a construção futura de novos módulos de teste para experimentação de outro tipo de vulnerabilidades.

Acunetix. (15 de Setembro de 2017). *Types of SQL Injection (SQLi).* Obtido de Acunetix: https://www.acunetix.com/websitesecurity/sql-injection2/

Alecrim, E. (24 de Agosto de 2017). *O que é firewall? - Conceito, tipos e arquiteturas*. Obtido de Info Wester: https://www.infowester.com/firewall.php

ANACOM. (14 de Agosto de 2017). Obtido de ANACOM: https://www.anacom.pt/render.jsp?contentId=1374434

AspenCore. (27 de Outubro de 2017). *How secure is AES against brute force attacks?* Obtido de EE Times - Connecting the Global Electronics Community: https://www.eetimes.com/document.asp?doc_id=1279619

Bakhoff, M. (5 de Janeiro de 2017). *Modular multiplicative inverse function in Python*. Obtido de Stackoverflow: https://stackoverflow.com/questions/4798654/modular-multiplicative-inverse-function-in-python

Boneh, D. (Fevereiro de 1999). Twenty Years of of Attacks on the RSA Cryptosystem. *Notices of the AMS, Volume 46, Number 2*.

Boucinha, F. d. (Outubro de 2011). A Survey of Cryptanalytic Attacks on RSA. *Instituto Superior Técnico, Universidade Técnica de Lisboa, Tese de Mestrado*.

Cisco. (24 de Agosto de 2017). *What Is a Firewall?* Obtido de Cisco Systems, Inc.: https://www.cisco.com/c/en/us/products/security/firewalls/what-is-a-firewall.html

Codecademy. (Maio a Outubro de 2017). *SQL*. Obtido de Codecademy: https://www.codecademy.com/catalog/language/sql

Cromwell, B. (28 de Outubro de 2017). *Cipher Strength and Key Length*. Obtido de Just Enough Cryptography: https://cromwell-intl.com/cybersecurity/crypto/cipher-strength.html

DigitalOcean, I. (17 de Junho de 2017). *How To Install Linux, Apache, MySQL, PHP (LAMP) stack on Ubuntu 16.04*. Obtido de Digital Ocean: https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-16-04#how-to-find-your-server-39-s-public-ip-address

Dougherty, C. (2012). *Practical Identification of SQL Injection Vulnerabilities.* Obtido de US-CERT - United States Computer Emergency Rediness Team: https://www.us-cert.gov/sites/default/files/publications/Practical-SQLi-Identification.pdf

Fergunson, N., Schneier, B., & Kohno, T. (2010). *CRIPTOGRAPHY ENGENEERING - Design Principals and Practical Applications.* Indianopolis, USA: Wiley Publishing, Inc.

GitHub. (Janeiro a Outubro de 2017). Obtido de GitHub: https://github.com/

InfoSecResources. (30 de Setembro de 2017). *Analyzing the Mod Security Logs*. Obtido de Infosec Institute: http://resources.infosecinstitute.com/analyzing-mod-security-logs/#gref

Kallin, J., & Valbuena, I. L. (16 de Junho de 2017). *A comprehensive tutorial on cross-site scripting - Part One: Overview*. Obtido de Excess XSS : https://excess-xss.com/

Katz, J., & Lindell, Y. (2008). *Introduction to Modern Cryptography.* USA: Chapman & Hall/CRC.

KhanAcademy. (27 de Outubro de 2017). *Noções básicas de SQL*. Obtido de Khan Academy: https://pt.khanacademy.org/computing/computer-programming/sql/sql-basics/v/welcome-to-sql

Leite, H. I. (2009). Ataques ao Sistema Criptográfico RSA. *Universidade de Aveiro, Departamento de Matemática, Tese de Mestrado*.

Mahajan, P., & Sachdeva, A. (2013). A Study of Encryption Algorithms AES, DES and RSA for Security. *Global Journal of Computer Science and Technology Network, Web & Security - Volume 13, Issue 15 Version 1.0*.

Mao, W. (2004). *Modern Cryptography, Theory & Practice.* U.S.A.: Hewllet-Packard Company - Prentice Hall.

Messerges, T. S., Dabbish, E. A., & Sloan, R. (1999). Power Analysis Attacks of Modular Exponetiation in Smartcards. Springer - Verlag Berlin Heidelberg.

Microsoft. (5 de Agosto de 2017). *Noções básicas da base de dados*. Obtido de Microsoft: https://support.office.com/pt-pt/article/No%C3%A7%C3%B5es-b%C3%A1sicas-da-base-de-dados-a849ac16-07c7-4a31-9948-3c8c94a7c204

Netsparker, L. (16 de 06 de 2017). *Simple DOM Based Cross-site Scripting Vulnerability Example*. Obtido de netsparker: https://www.netsparker.com/blog/web-security/dom-based-cross-site-scripting-vulnerability/

Oracle. (27 de Outubro de 2017). *Key Length and Encryption Strength*. Obtido de Oracle: https://docs.oracle.com/cd/E19424-01/820-4811/aakfw/index.html

Ordem dos Engenheiros. (Maio/Junho de 2017). Revista Ingenium. *Revist Ingénium, Serie II, Nº 159*.

Outsystem. (13 de Outubro de 2017). *Server-side Validation of Inputs.* Obtido de Outsystem: https://www.outsystems.com/learn/lesson/989/server-side-validation-of-inputs/

OWASP. (19 de Setembro de 2017). *Testing for SQL Injection (OTG-INPVAL-005)*. Obtido de OWASP: https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005)#Detection_Techniques

OWASP. (Maio a Outubro de 2017). *Welcome to QWASP - the free and open software security community*. Obtido de OWASP: https://www.owasp.org/index.php/Main_Page

Paar, C. (30 de Janeiro de 2014). Lecture 12: The RSA Cryptosystem and Efficient Exponentiation by Christof Paar. *Ruhr University Bochum, Germany - Winter semester, 2010*. Obtido de https://www.youtube.com/watch?v=QSlWzKNbKrU

Paar, C., & Pelzl, J. (2010). *Understanding Cryptography.* Springer.

Paola, S. d., & Carettoni, L. (31 de Outubo de 2017). *Testing for Reflected Cross site scripting*. Obtido de OWSAP: https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OTG-INPVAL-001)

pentestmonkey. (Agosto a Outubro de 2017). *MySQL SQL Injection Cheat Sheet*. Obtido de pentestmonkey: http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet

Petri, A. C. (5 de Dezembro de 2016). *falcaopetri/euclides*. Obtido de Github: https://github.com/falcaopetri/euclides/blob/master/euclides.py

PHPGroup. (Junho a Outubro de 2017). *PHP is a popular general-purpose scripting language that is especially suited to web development.* Obtido de PHP: http://php.net/

radx64. (2 de Maio de 2017). *radx64/rsa-wiener-attack*. Obtido de GitHub: https://github.com/radx64/rsa-wiener-attack/blob/master/main.py

Ristić, I. (2017). ModSecurity Handbook - Second Edition. Em I. Ristić. Feisty Duck. Obtido de https://www.feistyduck.com/library/modsecurity-handbook-free/online/ch04-logging.html

Salah, I. K., Abdullah, D., & Oqeilei. (2006). Mathematical Attacks on RSA Cryptosystem. *Journal of Computer Science*, 665-671.

Shah, S. (2002). *Top Ten Web Attacks.* Obtido de net square: https://www.blackhat.com/presentations/bh-asia-02/bh-asia-02-shah.pdf

Singh, S. (1999). *The Code Book.* London: Fourth Estate.

Sipser, M. (2013). *Introduction to the Theory of Computation (3rd editio).* India: Cengage Learning.

sqlinjection.net. (Agosto a Outubro de 2017). Obtido de sqlinjection.net: http://www.sqlinjection.net/

Stinson, D. R. (2005). *Cryptography. Theory and Practice (Third Edition).* CRC Press.

Stuttard, D., & Pinto, M. (2011). *The Web Application Hacker's Handbook.* Indianapolis, USA: Wiley Publishig, Inc.

Sucuri, I. (01 de Outubro de 2017). *Most Common Attacks Affecting Today's Websites*. Obtido de Sucuri Blog: https://blog.sucuri.net/2014/11/most-common-attacks-affecting-todays-websites.html

Trustwave Holdings, I. (29 de Setembro de 2017). *Mode Security - Open Source Application Firewall*. Obtido de Mode Security - Open Source Application Firewall: https://modsecurity.org/download.html

Trustwave Holdings, I. (30 de Setembro de 2017). *SpiderLabs/ModSecurity*. Obtido de Github: https://github.com/SpiderLabs/ModSecurity/wiki/ModSecurity-2-Data-Formats#Alert_Action_Description

Tutorialspoint. (13 de Junho de 2017). *PHP - MySQL Login*. Obtido de Tutorialspoint: https://www.tutorialspoint.com/php/php_mysql_login.htm

W3school. (Maio a Outubro de 2017). *SQL Tutorial*. Obtido de W3school.com: https://www.w3schools.com/sql/

Wiener, M. J. (Maio de 1990). Cryptanalysis of Short RSA Secret Exponents. *IEEE Transaction of Information Theory, vol 36, nº3*.

Zúquete, A. (2013). *Segurança em Redes Informáticas.* Lisboa: FCA - Editora de Informática.

Anexo

Algoritmo do ataque de Wiener

Ataque de Wiener

Ficheiro: **wiener-attack.py**

Adaptado do ataque de radx64, publicado em GitHub, em 11 de Junho de 2015 (radx64, 2017)

```
from fractions import Fraction
import math
from Gerador_Wiener import *
import socket
import thread
import time

def fractionToFloat(fraction):
    d = float(fraction)
    return d

def printFractions(fractions):
    for fraction in fractions:
      print "%f \t\t= \t\t%s" % (fraction, fraction)

def getContinuedFractions(fraction):
    x = fraction.numerator
    y = fraction.denominator
    if (x % y) == 0:
        return [x/y]
    fractions = getContinuedFractions(Fraction(y, x % y))
    fractions.insert(0, x / y)
    return(fractions)


def getFractionFromContinuedFractions(fractions):
    length = len(fractions)
    if length == 0:
        return(Fraction(0,1))
    elif length == 1:
        return(Fraction(fractions[0],1))
    else:
        rest = fractions[1:length]
        f = getFractionFromContinuedFractions(rest)
        return(Fraction(fractions[0]*f.numerator+f.denominator,
f.numerator))

def getConvergents(fractions):
    convergents = []
    for i, _ in enumerate(fractions):

convergents.append(getFractionFromContinuedFractions(fractions[0:i]))
    return convergents


def attack(e, N):
    continuedFraction = getContinuedFractions(Fraction(e,N))
```

```
    convergents = getConvergents(continuedFraction)

    for frac in convergents:
        de = e*frac.denominator
        if frac.numerator != 0 and (de-1)%frac.numerator == 0:
            fi = (de-1) // frac.numerator
            b = N - fi + 1
            delta = b*b - 4*N
            if delta>=0:
                x = math.sqrt(delta)
                if (x % 1 == 0) and (frac.denominator > 1):
                    d = frac.denominator
    return d

s = socket.create_connection(('localhost', '1234'))

time.sleep(1)
x = s.recv(2048).strip()
print x

y = (x.split('\n')[1].split(' '))

c = int(y[0])
e = int(y[1])
n = int(y[2])

d = attack(e,n)
x = pow(c,d,n)
print "[+] plaintext is = ", x
s.send(str(x))
print s.recv(2048)
```

Gerador de chaves para o ataque de Wiener

Ficheiro: **Gerador_Wiener.py**

(Bakhoff, 2017), (Petri, 2016)

```
import random
import math

def egcd(a, b):
 if (a == 0):
     return [b, 0, 1]
 else:
     g, y, x = egcd(b % a, a)
     return [g, x - (b // a) * y, y]

def modInv(a, m):
 g, x, y = egcd(a, m)
 if (g != 1):
     print '**************************************************'
     raise Exception("[-]No modular multiplicative inverse of %d under
modulus %d" % (a, m))
 else:
     return x % m

def decrypt(p, q, d, c):
```

```
      n = p * q
      phi = (p - 1) * (q - 1)
      m = pow(c, d, n)
      m = m%n
      print 'm=',m
      return m

def encrypt(p, q, e, m):
      n = p * q
      c = pow(m, e, n)
      c = c%n
      print 'c=',c
      return c

def teste(p,q,n,e,d,m,c):
      print 'primos-p_e_q =',primos
      print 'p =',p
      print 'q =',q
      print 'n =',n
      print 'phi =',phi
      print 'primos-d =',primos1
      print 'd =',d
      print 'e =',e
      print 'm_escolhido =',m,' m_decifrado = ',m1
      print 'c =',c

primos =
[1009,1013,1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,1087,1091
,1093,1097,1103,1109,1117,1123,1129,1151,1153,1163,1171,1181,1187,1193
,1201,1213,1217,1223,1229,1231,1237,1249,1259,1277,1279,1283,
1289,1291,1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,1381,1399,
1409,1423,1427,1429,1433,1439,1447,1451,1453,1459,1471,1481,1483,1487,
1489,1493,1499,1511,1523,1531,1543,1549,1553,1559,1567,1571,
1579,1583,1597,1601,1607,1609,1613,1619,1621,1627,1637,1657,1663,1667,
1669,1693,1697,1699,1709,1721,1723,1733,1741,1747,1753,1759,1777,1783,
1787,1789,1801,1811,1823,1831,1847,1861,1867,1871,1873,1877,
1879,1889,1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,1993,1997,
1999]

primos1 =
[3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]


def generate_challenge():
      while True:
            try:
                  p = random.choice (primos)
                  q = random.choice (primos)
                  assert q < p
                  assert p < 2*q

                  n = p*q
                  phi = (p-1)*(q-1)

                  d = random.choice (primos1)
                  e = modInv(d, phi)
                  r = math.sqrt(n)
                  r = math.sqrt(r)
                  assert 3*d < r
                  assert (e*d) % phi == 1
                  break
```

```
            except:
                  continue

      return n, e, d, p, q
```

Servidor do ataque de Wiener

Ficheiro: **server2_Wiener*.py***

```
import socket
import sys
import random
from thread import *
from Gerador_Wiener import *

HOST = 'localhost'
PORT = 1234

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'Socket created'

try:
      s.bind((HOST, PORT))
except socket.error as msg:
      print 'Bind failed. Error Code : ' + str(msg[0]) + ' Message ' +
msg[1]
      sys.exit()

print 'Socket bind complete'

s.listen(10)
print 'Socket now listening'

def clientthread(conn):
      conn.send('Welcome to our cipher scheme.Can you break it? (c, e,
n)\n')
      n, e, d, p, q = generate_challenge()


      tries=0
      while True:
            goal = random.randrange(2,n,1)
            print "[+] Tuples: n,e: ",n, e #, d, p, q
            print "[+] Message: ", goal
            to_send = str(pow(goal,e,n)) + ' ' + str(e) + ' ' + str(n)

            print "[+] Sending Challenge"
            conn.sendall(str(to_send)+'\n')
            conn.send('What is the message ?\n')

            m = conn.recv(1024).strip()
            print "[+] Received Answer   : ", m

            tries += 1
            if m == str(goal):
                  print "[+] CORRECT" # , tries
                  conn.sendall("Congrats. Correct.\n")
                  break
            else:
```

```
                  print "[-] Missed try ", tries
                  conn.sendall("Incorrect. Please try again. (try " +
str(tries) + ")\n")

      conn.close()

while 1:
      conn, addr = s.accept()
      print 'Connected with ' + addr[0] + ':' + str(addr[1])

      start_new_thread(clientthread ,(conn,))

s.close()
```

Algoritmo do ataque de Módulo Comum

Common Modulus Attack

Ficheiro: **common-modulus-attack.py**

Adaptado do ataque de abpolym, publicado em GitHub, em 11 de Junho de 2015

```
""" RSA Solving Module by polym / Tim https://github.com/abpolym """

import itertools, sys
from sage.all import *
import socket
from Gerador_Common_Modulus import *
import time

def common_modulus(N, cs, es):
      for (e1, e2) in itertools.combinations(es, 2):
            assert e1!=e2
            if gcd(e1, e2) != 1: continue
            g, x, y = xgcd(e1, e2)
            if g!=1: raise Exception("WTF?")
            E = cs[es.index(e1)]
            if x < 0:
                  E = inverse_mod(E, N)
                  x *= -1
            F = cs[es.index(e2)]
            if y < 0:
                  F = inverse_mod(F, N)
                  y *= -1
            p1 = pow(E, x, N)
            p2 = pow(F, y, N)
            return (p1*p2) % N

s = socket.create_connection(('localhost', '1235'))

time.sleep(3)
x = s.recv(2048).strip()
print x

y = (x.split('\n')[1].split(' '))

n = int(y[0])
print "n =", n
c1 = int(y[1])
print "c1 =", c1
c2 = int(y[2])
print "c2 =", c2
e1 = int(y[3])
print "e1 =", e1
e2 = int(y[4])
print "e2 =", e2

cs = [c1,c2]
es = [e1,e2]

y = common_modulus(n, cs, es)
print "m =", y

s.send(str(y)+'\n')
print s.recv(2048)
```

Gerador de chaves para o Common Modulus Attack

Ficheiro: **Gerador_Common_Modulus.py**

```
import random
import math

def egcd(a, b):
 if (a == 0):
     return [b, 0, 1]
 else:
     g, y, x = egcd(b % a, a)
     return [g, x - (b // a) * y, y]

def modInv(a, m):
 g, x, y = egcd(a, m)
 if (g != 1):
     raise Exception("[-]No modular multiplicative inverse of %d under
modulus %d" % (a, m))
 else:
     return x % m

def decrypt(p, q, d, c):
      n = p * q
      print 'n=',n
      phi = (p - 1) * (q - 1)
      m = pow(c, d, n)
      m = m%n
      return m

def encrypt(p, q, e, m):
      n = p * q
      c = pow(m, e, n)
      c = c%n
      return c

primos =
[1009,1013,1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,1087,1091
,1093,1097,1103,1109,1117,1123,1129,1151,1153,1163,1171,1181,1187,1193
,1201,1213,1217,1223,1229,1231,1237,1249,1259,1277,1279,1283,
1289,1291,1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,1381,1399,
1409,1423,1427,1429,1433,1439,1447,1451,1453,1459,1471,1481,1483,1487,
1489,1493,1499,1511,1523,1531,1543,1549,1553,1559,1567,1571,
1579,1583,1597,1601,1607,1609,1613,1619,1621,1627,1637,1657,1663,1667,
1669,1693,1697,1699,1709,1721,1723,1733,1741,1747,1753,1759,1777,1783,
1787,1789,1801,1811,1823,1831,1847,1861,1867,1871,1873,1877,
1879,1889,1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,1993,1997,
1999]
primos1 =
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,9
7]

def generate_challenge():
      while True:
            try:
                  p = random.choice (primos)
                  q = random.choice (primos)

                  n = p*q
```

```
                phi = (p-1)*(q-1)

                e1 = random.choice (primos1)

                e2 = random.choice (primos1)

                d1 = modInv(e1, phi)
                y1 = e1*d1
                z1 = y1%phi

                d2 = modInv(e2, phi)
                y2 = e2*d2
                z2 = y2%phi

                m = random.randint(2,n)
                # print "m = ",m

                c1 = encrypt(p, q, e1, m)
                c2 = encrypt(p, q, e2, m)

                break
          except:
                continue
    return n, e1, e2
```

Servidor do Common Modulus Attack

Ficheiro: **server2_CMA.*py***

```
import socket
import sys
import random
from thread import *
from Gerador_Common_Modulus import *


HOST = 'localhost'
PORT = 1235

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'Socket created'

try:
      s.bind((HOST, PORT))
except socket.error as msg:
      print 'Bind failed. Error Code : ' + str(msg[0]) + ' Message ' +
msg[1]
      sys.exit()

print 'Socket bind complete'

s.listen(10)
print 'Socket now listening'

def clientthread(conn):
      conn.send('Welcome to our cipher scheme.Can you break it? (c, e,
n)\n')
      n, e1, e2 = generate_challenge()
```

```
	tries=0
	while True:
		goal = random.randrange(2,n,1)
		print "[+] Tuples: n,e1,e2 ",n, e1, e2
		print "[+] Goal: ", goal
		to_send = str(n) + ' ' + str(pow(goal,e1,n)) + ' ' +
str(pow(goal,e2,n)) + ' ' + str(e1) + ' ' + str(e2)

		print "[+] Sending Challenge"
		conn.sendall(str(to_send)+'\n')
		conn.send('What is the message ?\n')

		m = conn.recv(2048).strip()
		print "[+] Received Answer   : ", m

		tries += 1
		if m == str(goal):
			print "[+] CORRECT " #, tries
			conn.sendall("Congrats. Correct.\n")
			break
		else:
			print "[-] Missed try ", tries
			conn.sendall("Incorrect. Please try again. (try " +
str(tries) + ")\n")

	conn.close()

while 1:
	conn, addr = s.accept()
	print 'Connected with ' + addr[0] + ':' + str(addr[1])

	start_new_thread(clientthread ,(conn,))

s.close()
```

Algoritmo do Least Significant Bit Attack

Least Significant Bit Attack

Ficheiro: **LSB-Attack.py**

```
# Binary Searching RSA plaintext using a parity Oracle
# Created on Jan 28, 2017
from math import log, ceil, floor, trunc
import itertools, sys
from sage.all import *
import socket
import time

s = socket.create_connection(('localhost', '1236'))

time.sleep(1)

x = s.recv(2048).strip()

y = (x.strip('\n').split(' '))

n = int(y[0])
print "n = ",n
c = int(y[1])
print "c = ",c
e = int(y[2])
print "e = ",e

a = 0
b = n
print '[',a,b,']'
for i in range (int(log(n,2)+2)) :
      if (b-a) >= 1:
            c= (pow(2,e,n)) * c %n
            f = 0
            print c
            to_send = str(c) + ' ' + str(f)
            s.sendall(str(to_send))

            paridade = s.recv(2048).strip()
            par = paridade.strip()
            pr = int(par[0])


            if pr ==0:
                  a = a
                  b = 1.0*(b - (b-a)/2)
                  print "i = ", i, "  parity ", paridade,
'[',ceil(a),floor(b),']'
            else:
                  a = 1.0*(a + (b-a)/2)
                  b = b
                  print "i = ", i, "  parity ", paridade,
'[',ceil(a),floor(b),']'

      else:
            m = int(b)
            f = 1
            to_send = str(m) + ' ' + str(f)
            s.send(to_send+'\n')
```

```
            break
```

Gerador de chaves para o LSB Attack

Ficheiro: **Gerador_LSB.py**

```
import random
import math

def egcd(a, b):
 if (a == 0):
     return [b, 0, 1]
 else:
     g, y, x = egcd(b % a, a)
     return [g, x - (b // a) * y, y]

def modInv(a, m):
 g, x, y = egcd(a, m)
 if (g != 1):
     raise Exception("[-]No modular multiplicative inverse of %d under
modulus %d" % (a, m))
 else:
     return x % m

def decrypt1(p, q, d, c):
      n = p * q
      phi = (p - 1) * (q - 1)
      m = pow(c, d, n)
      m = m%n
      return m

def decrypt(n, d, c):
      m = pow(c, d, n)
      return m

def encrypt(p, q, e, m):
      n = p * q
      c = pow(m, e, n)
      c = c%n
      return c

primos =
[1009,1013,1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,1087,1091
,1093,1097,1103,1109,1117,1123,1129,1151,1153,1163,1171,1181,1187,1193
,1201,1213,1217,1223,1229,1231,1237,1249,1259,1277,1279,1283,
1289,1291,1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,1381,1399,
1409,1423,1427,1429,1433,1439,1447,1451,1453,1459,1471,1481,1483,1487,
1489,1493,1499,1511,1523,1531,1543,1549,1553,1559,1567,1571,
1579,1583,1597,1601,1607,1609,1613,1619,1621,1627,1637,1657,1663,1667,
1669,1693,1697,1699,1709,1721,1723,1733,1741,1747,1753,1759,1777,1783,
1787,1789,1801,1811,1823,1831,1847,1861,1867,1871,1873,1877,
1879,1889,1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,1993,1997,
1999]
primos1 =
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,9
7]


def generate_challenge():
```

```
	while True:
		try:
			p = random.choice (primos)
			q = random.choice (primos)

			n = p*q

			phi = (p-1)*(q-1)

			e = random.choice (primos1)

			d = modInv(e, phi)
			y = e*d
			z = y%phi

			break
		except:
			continue
	return n, e, d
```

Servidor do LSB Attack

Ficheiro: **server2_LSB.*py***

```
import socket
import sys
import random
from thread import *
from Gerador_LSB import *
import time


HOST = 'localhost'
PORT = 1236

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'Socket created'

try:
	s.bind((HOST, PORT))
except socket.error as msg:
	print 'Bind failed. Error Code : ' + str(msg[0]) + ' Message ' +
msg[1]
	sys.exit()

print 'Socket bind complete'

s.listen(10)
print 'Socket now listening'

def oraculo(n, d, c1):
	z = decrypt(n,d,c1)%2
	return z

def clientthread(conn):
	n, e, d = generate_challenge()

	tries=0
```

```
	while True:
		goal = random.randrange(2,n,1)
		c = pow(goal,e,n)
		print 'cyphertext = ',c
		print "[+] Tuples: n,c,e", n, c, e
		print "[+] Goal: ", goal
		to_send = str(n) + ' ' + str(c) + ' ' + str(e)

		print "[+] Sending Challenge"
		conn.sendall(str(to_send)+'\n')

		while True:

			time.sleep(1)
			x = conn.recv(2048).strip()
			y = (x.strip('\n').split(' '))
			c1 = int(y[0])
			print "[+] Received Answer   : ", c1
			f = int(y[1])
			if f == 1:
				break
			else:
				paridade = oraculo(n, d, c1)
				print paridade
				to_send = str(paridade)
				conn.sendall(str(to_send)+'\n')

		m = str(c1)
			print "[+] plaintext  : ", m

		tries += 1
		if m == str(goal):
			print "[+] CORRECT "
			conn.sendall("Congrats. Correct.\n")
			print
"************************************************************"
			break
		else:
			print "[-] Missed try ", tries
			conn.sendall("Incorrect. Please try again. (try " +
str(tries) + ")\n")

	conn.close()

while 1:
	conn, addr = s.accept()
	print 'Connected with ' + addr[0] + ':' + str(addr[1])

	start_new_thread(clientthread ,(conn,))

s.close()
```

Algoritmos do WEB attacks e WEB attacks + Mod Security

Base de dados (Mysql)

id	NAME	PASSWORD	BIRTHDAY
1	Vasco	vasco123	2003-09-05
2	Vera	vera123	2003-09-05
3	Bernardo	bernardo123	2000-06-14
4	Tomas	tomas123	1998-05-20
5	Carmo	carmo12	2000-10-30
6	Pilar	pilar123	2008-11-14
7	Joao	joao123	2006-12-12
8	Leonor	leonor123	2002-11-18
9	Joaquim	joaquim123	2005-01-07
10	Madalena	madalena123	1996-11-12
11	Antonio	antonio123	2000-11-01
12	Francisca	francisca123	2003-09-17
13	Vicente	vicente123	2011-04-06
14	Mafalda	mafalda123	1995-12-16
15	Maria	maria23	1998-02-06
16	Tiago	tiago123	1994-10-28
17	Francisco	francisco123	1998-10-26

Tabela familia

id	Brand	Model	Type
1	Alhambra	5P CW E2	classic
2	Gibson	Les Paul	electric

Tabela guitarras

Ficheiro **config.php**

```
<?php

      define("DB_SERVER", "localhost");
        define("DB_USERNAME", "root");
        define("DB_PASSWORD", "reverse");
        define("DB_DATABASE", "family");

      $db =
mysqli_connect(DB_SERVER,DB_USERNAME,DB_PASSWORD,DB_DATABASE) or
die("ERRO" . mysqli_error($db));

?>
```

Ficheiro **Login.php**

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>Login Page</title>

    <style type="text/css">
        body {
            font-family: Arial, Helvetica, sans-serif;
            font-size: 14px;
        }

        label {
            font-weight: bold;
            width: 100px;
            font-size: 14px;
        }

        .box {
            border: #18C5DA solid 1px;
        }
    </style>

</head>

<h1>WEB ATTACKS</h1>
</html>

<?php

    include('Config.php');


      if($_SERVER["REQUEST_METHOD"] == "POST")
      {
            echo "<HR>";
            echo DB_SERVER;
            echo "<HR><HR>";

      if(mysqli_connect_error())
      {
            echo mysqli_connect_error();
      }

            echo "<br>";
        $myusername = $_REQUEST["NAME"];
        $mypassword = $_REQUEST['PASSWORD'];
            echo "<BR>";
            echo "$myusername" ;
            echo "<BR>";
            echo "$mypassword" ;
            echo "<HR>";

        $sql = "SELECT * FROM family.familia WHERE NAME =
'$myusername' and PASSWORD = '$mypassword'";
            echo $sql;
            echo "<BR>";
        $result = mysqli_query($db,$sql);
        $row = mysqli_fetch_array($result,MYSQLI_ASSOC);
        $active = $row['active'];
        $count = mysqli_num_rows($result);
            echo "<BR>Numero de linhas que verificam a query = ";
            echo $count;
```

```
      if($count != 0)
      {
            session_start();
            $_SESSION["myusername"] = $myusername;
            $user_check = $_SESSION['myusername'];
                  $ses_sql = mysqli_query($db,"select NAME from
family.familia where NAME = ' $user_check ' ");
            $row = mysqli_fetch_array($ses_sql,MYSQLI_ASSOC);
            $login_session = $row['NAME'];

            mysqli_close($db);

            if(isset($_SESSION["myusername"]))
            {

      $link="<script>window.open('welcome.php?User=$login_session');</
script>";
                        echo $link;
            }
            else
            {
                  echo "<BR>NÃ£o autorizado";
            }
      }
      else
      {
             $error = "Your Login Name or Password is invalid";
      }
   }
?>

<html>
<body bgcolor="#FFFFFF">

    <div align="center">
        <div style="width:370px; border: solid 1px #9CA7A8; "
align="left">
            <div style="background-color:#18C5DA; color:#FFFFFF;
padding:3px;"><b>Login</b></div>

            <div style="margin:30px">

                <form action="" method="post">
                    <label>UserName  :</label><input type="text"
name="NAME" class="box" /><br /><br />
                    <label>Password  :</label><input type="password"
name="PASSWORD" class="box" /><br /><br />
                    <input type="submit" value=" Submit " /><br />
                </form>

                <div style="font-size:11px; color:#cc0000; margin-
top:10px"><?php echo $error; ?></div>

            </div>

        </div>

    </div>

</body>
```

```
</html>
```

Ficheiro **welcome.php**

```
<?php
    include('Config.php');
    session_start();

      $name = $_SESSION['myusername'];

      $wel_sql = "SELECT * FROM family.familia WHERE NAME = '$name'";
            echo "<br>";
      $resultado = mysqli_query($db,$wel_sql);
?>

<html>

   <head>
      <title>Welcome </title>
   </head>

   <body>
      <h1>Welcome <?php echo $_SESSION['myusername']; ?></h1>
      <h3>Os seus dados</h3>

        <?php
      while($row = mysqli_fetch_array($resultado))
      {
            echo $row[id]."  ---  ".$row[NAME]."  ---
".$row['PASSWORD']."  ---  ".$row['BIRTHDAY']."<br>";
            echo "<br>";
      }
        ?>

      <h2><a href = "Logout.php">Sign Out</a></h2>
   </body>

</html>
```

Ficheiro **Logout.php**

```
<?php
   session_start();

   if(session_destroy()) {
      header("Location: Login.php");
   }
?>
```

(Tutorialspoint, 2017)

Resultados do ficheiro do Mod Security que audita a entrada de dados

Excerto do **modsec_audit.log**

Resposta ao ataque: **' or true -- -**

[**11/Oct/2017:07:23:47** +0100] Wd24838AAQEAAE8R4s8AAAAE **127.0.0.1** 42440 **127.0.0.1 80**

--a0b60525-B--

POST /Login.php HTTP/1.1

Host: localhost

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:51.0) Gecko/20100101 Firefox/51.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: http://localhost/Login.php

Cookie: PHPSESSID=7hfn03gkoeghqbtvu3rpqfhlo4

Connection: keep-alive

Upgrade-Insecure-Requests: 1

Content-Type: application/x-www-form-urlencoded

Content-Length: 31

--a0b60525-C--

NAME=%27+or+true+--+-&PASSWORD=

--a0b60525-F--

HTTP/1.1 403 Forbidden

Content-Length: 293

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: text/html; charset=iso-8859-1

--a0b60525-E--

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html><head>

<title>403 Forbidden</title>

</head><body>

<h1>Forbidden</h1>

You don't have permission to access /Login.php

on this server.

<hr>

<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>

</body></html>

--a0b60525-H--

Message: Warning. detected SQLi using libinjection with fingerprint 's&1c' [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "68"] [id "942100"] [rev "1"] [msg "SQL Injection Attack Detected via libinjection"] [data "Matched Data: s&1c found within ARGS:NAME: ' or true -- -"] [severity "CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity "1"] [accuracy "8"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"]

Message: Warning. detected SQLi using libinjection with fingerprint 's&1' [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "68"] [id "942100"] [rev "1"] [msg "SQL Injection Attack Detected via libinjection"] [data "Matched Data: s&1 found within ARGS:NAME: ' or true "] [severity "CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity "1"] [accuracy "8"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"]

Message: Access denied with code 403 (phase 2). Operator GE matched 5 at TX:anomaly_score. [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "57"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Total Score: 10)"] [severity "CRITICAL"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-generic"]

Message: Warning. Operator GE matched 5 at TX:inbound_anomaly_score. [file "/etc/apache2/owasp-modsecurity-crs/rules/RESPONSE-980-CORRELATION.conf"] [line "73"] [id "980130"] [msg "Inbound Anomaly Score Exceeded (Total Inbound Score: 10 - **SQLI=10**,XSS=0,RFI=0,LFI=0,RCE=0,PHPI=0,HTTP=0,SESS=0): **SQL Injection Attack Detected** via libinjection"] [tag "event-correlation"]

Action: Intercepted (phase 2)

Apache-Handler: application/x-httpd-php

Stopwatch: 1507703027910545 21189 (- - -)

Stopwatch2: 1507703027910545 21189; combined=14426, p1=10421, p2=3939, p3=0, p4=0, p5=66, sr=19, sw=0, l=0, gc=0

Response-Body-Transformed: Dechunked

Producer: ModSecurity for Apache/2.9.0 (http://www.modsecurity.org/); OWASP_CRS/3.0.2.

Server: Apache/2.4.18 (Ubuntu)

Engine-Mode: "ENABLED"

Resposta ao ataque: **' OR true ORDER BY NAME; -- -**

--a0b60525-Z--

[11/Oct/2017:07:25:31 +0100] Wd25W38AAQEAAE8NsB0AAAAA **127.0.0.1 42452 127.0.0.1 80**

--d4457f1f-B--

POST /Login.php HTTP/1.1

Host: localhost

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:51.0) Gecko/20100101 Firefox/51.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: http://localhost/Login.php

Cookie: PHPSESSID=7hfn03gkoeghqbtvu3rpqfhlo4

Connection: keep-alive

Upgrade-Insecure-Requests: 1

Content-Type: application/x-www-form-urlencoded

Content-Length: 48

--d4457f1f-C--

NAME=%27**+OR+TRUE+ORDER+BY+NAME**%3B+--+-&PASSWORD=

--d4457f1f-F--

HTTP/1.1 403 Forbidden

Content-Length: 293

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: text/html; charset=iso-8859-1

--d4457f1f-E--

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">

<html><head>

<title>403 Forbidden</title>

</head><body>

<h1>Forbidden</h1>

You don't have permission to access /Login.php

on this server.

<hr>

<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>

</body></html>

--d4457f1f-H--

Message: Warning. detected SQLi using libinjection with fingerprint 's&1Bn' [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "68"] [id "942100"] [rev "1"] [msg "SQL Injection Attack Detected via libinjection"] [data "Matched Data: s&1Bn found within ARGS:NAME: ' OR TRUE ORDER BY NAME; -- -"] [severity "CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity "1"] [accuracy "8"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"]

Message: Warning. detected SQLi using libinjection with fingerprint 's&1Bn' [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "68"] [id "942100"] [rev "1"] [msg "SQL Injection Attack Detected via libinjection"] [data "Matched Data: s&1Bn found within ARGS:NAME: ' OR TRUE ORDER BY NAME; "] [severity "CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity "1"] [accuracy "8"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag "WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"] [tag "PCI/6.5.2"]

Message: Access denied with code 403 (phase 2). Operator GE matched 5 at TX:anomaly_score. [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "57"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Total Score: 10)"] [severity "CRITICAL"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-generic"]

Message: Warning. Operator GE matched 5 at TX:inbound_anomaly_score. [file "/etc/apache2/owasp-modsecurity-crs/rules/RESPONSE-980-CORRELATION.conf"] [line "73"] [id "980130"] [msg "Inbound Anomaly Score Exceeded (Total Inbound Score: 10 -

SQLI=10,XSS=0,RFI=0,LFI=0,RCE=0,PHPI=0,HTTP=0,SESS=0): **SQL Injection Attack Detected via libinjection**"] [tag "event-correlation"]

Action: Intercepted (phase 2)

Apache-Handler: application/x-httpd-php

Stopwatch: 1507703131694615 1863 (- - -)

Stopwatch2: 1507703131694615 1863; combined=1429, p1=265, p2=1106, p3=0, p4=0, p5=58, sr=10, sw=0, l=0, gc=0

Response-Body-Transformed: Dechunked

Producer: ModSecurity for Apache/2.9.0 (http://www.modsecurity.org/); OWASP_CRS/3.0.2.

Server: Apache/2.4.18 (Ubuntu)

Engine-Mode: "ENABLED"

--d4457f1f-Z—

Resposta ao ataque: **‘ OR TRUE AND SLEEP(1); -- -**

--9be53120-A--

[11/Oct/2017:07:26:40 +0100] Wd25oH8AAQEAAE8PgScAAAAC 127.0.0.1 42456 127.0.0.1 80

--9be53120-B--

POST /Login.php HTTP/1.1

Host: localhost

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:51.0) Gecko/20100101 Firefox/51.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: http://localhost/Login.php

Cookie: PHPSESSID=7hfn03gkoeghqbtvu3rpqfhlo4

Connection: keep-alive

Upgrade-Insecure-Requests: 1

Content-Type: application/x-www-form-urlencoded

Content-Length: 51

--9be53120-C--

NAME=%27+OR+TRUE+AND+SLEEP%281%29%3B+--+-&PASSWORD=

```
--9be53120-F--
HTTP/1.1 403 Forbidden
Content-Length: 293
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1

--9be53120-E--
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access /Login.php
on this server.<br />
</p>
<hr>
<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
</body></html>

--9be53120-H--
Message: Warning. detected SQLi using libinjection with fingerprint
's&f(1' [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-942-
APPLICATION-ATTACK-SQLI.conf"] [line "68"] [id "942100"] [rev "1"] [msg
"SQL Injection Attack Detected via libinjection"] [data "Matched Data:
s&f(1 found within ARGS:NAME: ' OR TRUE AND SLEEP(1); -- -"] [severity
"CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity "1"] [accuracy "8"] [tag
"application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag
"attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag
"WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"]
[tag "PCI/6.5.2"]
Message: Warning. detected SQLi using libinjection with fingerprint
's&f(1' [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-942-
APPLICATION-ATTACK-SQLI.conf"] [line "68"] [id "942100"] [rev "1"] [msg
"SQL Injection Attack Detected via libinjection"] [data "Matched Data:
s&f(1 found within ARGS:NAME: ' OR TRUE AND SLEEP(1); "] [severity
"CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity "1"] [accuracy "8"] [tag
"application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag
"attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"] [tag
"WASCTC/WASC-19"] [tag "OWASP_TOP_10/A1"] [tag "OWASP_AppSensor/CIE1"]
[tag "PCI/6.5.2"]
```

Message: Warning. Pattern match "(?i:(sleep\\((\\s*?)(\\d*?)(\\s*?)\\)|benchmark\\((.*?)\\,(.*?)\\)))" at ARGS:NAME. [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-942-APPLICATION-ATTACK-SQLI.conf"] [line "127"] [id "942160"] [rev "2"] [msg "Detects blind sqli tests using sleep() or benchmark()."] [data "Matched Data: SLEEP(1) found within ARGS:NAME: ' OR TRUE AND SLEEP(1); -- -"] [severity "CRITICAL"] [ver "OWASP_CRS/3.0.0"] [maturity "9"] [accuracy "8"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-sqli"] [tag "OWASP_CRS/WEB_ATTACK/SQL_INJECTION"]

Message: Access denied with code 403 (phase 2). Operator GE matched 5 at TX:anomaly_score. [file "/etc/apache2/owasp-modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "57"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Total Score: 15)"] [severity "CRITICAL"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-generic"]

Message: Warning. Operator GE matched 5 at TX:inbound_anomaly_score. [file "/etc/apache2/owasp-modsecurity-crs/rules/RESPONSE-980-CORRELATION.conf"] [line "73"] [id "980130"] [msg "Inbound Anomaly Score Exceeded (Total Inbound Score: 15 - SQLI=15,XSS=0,RFI=0,LFI=0,RCE=0,PHPI=0,HTTP=0,SESS=0): Detects blind sqli tests using sleep() or benchmark()."] [tag "event-correlation"]

Action: Intercepted (phase 2)

Apache-Handler: application/x-httpd-php

Stopwatch: 1507703200388550 2033 (- - -)

Stopwatch2: 1507703200388550 2033; combined=1664, p1=328, p2=1279, p3=0, p4=0, p5=57, sr=12, sw=0, l=0, gc=0

Response-Body-Transformed: Dechunked

Producer: ModSecurity for Apache/2.9.0 (http://www.modsecurity.org/); OWASP_CRS/3.0.2.

Server: Apache/2.4.18 (Ubuntu)

Engine-Mode: "ENABLED"

--9be53120-Z--