RSA Cryptography Algorithm: An Impressive Tool in Decreasing Intrusion Detection System Vulnerabilities in Network Security

S.H.Mortazavi, P.S.Avadhani

Abstract -This paper is discussed the security of computer networks is a concern for businesses and individuals who are aware of its advantages due to its flexibility. With the increase security of IDS for companies and homes, where information property are shared continually, security is of the nature.. Cryptography is seen as a major instrumentation in the line of defense of network security. This paper discusses the various RSA cryptography algorithm used in network security especially IDS and how effective they are in keeping IDS secure. The risks of using this algorithm are Specified and recommendations for securing IDS are reviewed.

Keywords- IDS,NIDS, HIDS, Encryption, Cryptography, RSA, DoS.

I. INTRODUCTION

Intrusion Detection Systems (IDS) are software or hardware products that automate the analysis process of traffic on a network or a host. And they are a complementary security tool in computer networks; it can be deployed at different points depending on the

application, host or network segment to be monitored. Accordingly to its location, the IDS must be parameterized in a different way, for example, an IDS located in a Demilitarized Zone (DMZ) must be more flexible than an IDS located inside the internal network to reduce false alarms or to avoid system overload, allowing intrusions without generating an alarm. Likewise the IDS can receive different kinds of attacks if it is located in a DMZ or in the intranet zone. Due to the increasing rate of attacks, Intrusion Detection Systems has become a complementary and mandatory security tool to any organization, and in addition it is useful to perform forensic analysis procedures in order to complement the IDS use. An IDS performs passive monitoring and captures information to be analyzed subsequently, it can launch an alarm to a server or send an email warning about possible intrusions but it cannot modify its environment, otherwise it is named Intrusion Prevention System (IPS). An IPS responds in real time if an intrusion is

Detected, the IPS takes an action modifying its environment; it could modify the firewall by closing a suspicious connection, or reconfigure the router, etc. In the last two decades many research studies about technologies, architectures, methodologies and technologies have been proposed in order to increase the IDS effectiveness.

Manuscript received on March, 2013.

S.H.Mortazavi, Department Of Computer Science and System Engineering, Andhra University, India.

Prof.P.S.Avadhani, Department Of Computer Science and System Engineering, Andhra University, India.

IDS are available in many different types and will approach the mission of uncovering shady traffic in Various ways. You can find host-based (HIDS) and network-based (NIDS) systems. Additionally, there are also IDS which detect movements by searching for particular signatures of well-known threats, just like the way how antivirus software generally detects and safeguards against malware and also IDS which detect by assessing traffic patterns against the baseline and look for anomalies. Besides that, there are that basically observe and alert, plus systems that carry out an action or even actions in Reaction to a recognized threat. Hackers can Permeate IDS software because some of its weaknesses. Figure 1 shown attack detection by an intrusion detection system:

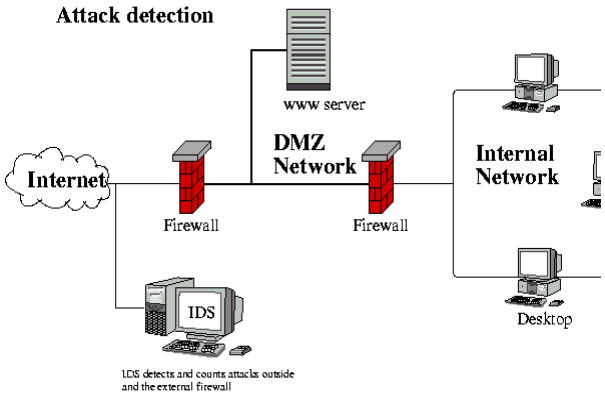

Figure 1: attack detection by an intrusion detection system

II. GENERAL ATTACKS

There are numerous network-based attacks that can be found in both wired and wireless network, this first part sums up the main ones.

- ***IP Spoofing***

The attacker uses the IP address of another machine to be ”hidden” and exploit weak authentication methods. If the target machines rely on the IP address to authenticate (e.g.source IP address access lists), IP spoofing can give an attacker access to systems he shouldnot have access to. Additionally, IP spoofing can make it very difficult to apprehend an attacker because logs will contain decoy addresses instead of real ones.

- ***Session hijack***

The attack consists in stealing network connections by kicking off the legitimate user or sharing a login. This type of attack is used against services with persistent login sessions, such as Telnet, rlogin, or FTP.

For any of these services, an attacker can hijack a session and cause a great amount of damage. The connection between two hosts is monitored to observe the TCP sequence number of the packet. The attacker guesses the appropriate sequence numbers and spoofs one of the participants address, taking over the conversation with the other participant.

- ***Denial-of-service (DoS)***

DoS attacks are among the most common exploits available today. As their name implies, a denial-of-service attack prevents (legitimate) users from being able to use a service. By bringing down critical servers, these attacks could also present a real physical threat to life. An attacker can cause the denial of service by flooding a system with bogus traffic. The technical result of a denial of service can range from applications sending wrong results to machines being down. Malformed Packet Attacks can be used to make DoS by generated badly formatted packets. Many vendor product implementations do not take into account all variations of user entries or packet types. If the software handles such errors poorly, the system may crash when receiving such packets. A classic example of this type of attack Involves sending IP fragments to a system that overlap with each other. Some unpatched Windows and Linux systems will crash when they encounter such packets.

- ***Buffer overflow***

Buffer overflows attacks consists in injecting data into a program (e.g. in web page form) to run malicious code. They can be used by an attacker to take control of a remote system or by local malicious users to elevate their privileges. This exploits weak parameters verifications. If user input length is not examined by the code, a particular variable on the stack may exceed the memory allocated to it on the stack, overwriting all variables and even the return

address for where execution should resume after the function is complete. Therefore, with very carefully constructed code, the attacker can actually enter information as a user into a program that consists of executable code and a new return address. Such attack allows an attacker to break out of the application code, and access any system components with the permissions of the broken program. If the broken program is running with super user privileges, the attacker has taken over the machine with a buffer overflow.

III. PUBLIC KEY CRYPTOGRAPHY

The concept of public key cryptography evolved from an attempt to attack two of the most difficult problems associated with symmetric encryption. Public key cryptography is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality. [1]

A public key encryption scheme has six ingredients the following characteristic:

- **Plaintext:** this is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** performs various Transfer matins on the plaintext.
- **Public and private key:** this is a pair of keys that have been selected so that if one is used for encryption, the other used for decryption.
- **Cipher text**: this is a scrambled message produced as output.
- **Decryption algorithm**: this algorithm accepts the cipher text and the matching key and produces the original plaintext[1].

3.1 RSA

One of the first successful responses to the challenge was developed in 1977 by Ron Rivest, Adi Shamir, and Len Adleman at MIT and first published in 1978.

The Rivest-Shamir-Adleman (RSA) scheme has since that time reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption.

The RSA scheme is a block cipher in which the plaintext and cipher text are integers between 0 and n- 1 for some n. A typical size for n is 1024 bits, or 309 decimal digits. That is, n is less than 2^1024.We examine RSA in this section in some detail, beginning with an explanation of the algorithm. Then we examine some of the computational and cryptanalytic implications of RSA [1].

3.1.1 Encryption and Decryption of RSA algorithm

RSA makes use of an expression with exponentials. Plaintext is encrypted in blocks, with each block having a binary value less than some number n. that is, the block size must be less than or equal to **log2(n) + 1**; in practice, the block size is i bits, where **2^i<n≤2^i+1**. Encryption and decryption are of the following form, for some plaintext block M and cipher text block C.[1]

$$C = M^e \bmod n$$

$$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Both sender and receiver must know the value of n. The sender knows the value of e, and only the receiver knows the value of d. Thus, this is a public-key encryption algorithm with a public key of **PU = {e, n}** and a private key of **PR = {d, n}**.

For this algorithm to be satisfactory for public-key encryption, the following requirements must be met.

1. It is possible to find values of e, d, n such that **M^ed mod n = M** for all M < n.
2. It is relatively easy to calculate **M^e mod n** and

C^d mod n for all values of M < n.

3. It is infeasible to determine d given e and n.

For now, we focus on the first requirement and consider the other questions

later. We need to find a relationship of the form.

M^ed mod n = M

The preceding relationship holds if e and d are multiplicative inverses modulo **φ(n)**, where **φ(n)** is the Euler totient function.

q prime, **φ (pq) = (p - 1)(q - 1)**. The relationship between e and d can be

expressed as:

$$e^d \bmod \varphi(n) = 1$$

This is equivalent to saying:

$$ed = 1 \bmod \varphi(n)$$

$$d = e^{-1} \bmod \varphi(n)$$

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

That is, e and d are multiplicative inverses mod **φ (n)**. Note that, according to the rules of modular arithmetic, this is true only if d (and therefore e) is relatively prime to **φ(n).**

Equivalently, **gcd(φ(n), d) = 1**. Satisfies the requirement for RSA. We are now ready to state the RSA scheme. The ingredients are the following:

p, q, two prime numbers	(private, chosen)
n = pq	(public, calculated)
***e*, with gcd(φ(n),e)=1;1<e< φ(n)**	(public,chosen)
d=e^-1(mod φ(n))	(private, calculated)

The private key consists of **{*d, n*}** and the public key consists of {*e, n*}. Suppose that user **A** has published its public key and that user **B** wishes to send the message **M** to **A**. Then **B** calculates ***C* = *M*^*e* mod *n*** and transmits *C*. On receipt of this cipher text,

User **A** decrypts by calculating ***M* = *C*^*d* mod *n***.[1]

3.1.2 Computational Aspects

We now turn to the issue of the complexity of the computation required to use Rather are actually two issues to consider: encryption/decryption and key generation. Let us look first at the process of encryption and decryption and then consider

Key generation[1].

3.1.3 Exponentiation In Modular Arithmetic

Both encryption and decryption in **RSA** involve raising an integer to an integer power, mod *n*. If the exponentiation is done over the integers and then reduced modulo *n*, the intermediate values would Be gargantuan. Fortunately, as the preceding example shows, we can make use of a property of modular arithmetic:

[(*a* mod *n*) * (*b* mod *n*)] mod *n* = (*a* * *b*) mod *n*

Thus, we can reduce intermediate results modulo *n*. This makes the calculation practical.

Another consideration is the efficiency of exponentiation, because with **RSA**, we are dealing with potentially large exponents. To see how efficiency might be increased, consider that we wish to compute *x*^16. A straightforward approach requires 15 multiplications:

X*^16 = *x* * *x* * *x* * *x* * *x* * *x* * *x* * *x* * *x* * *x* * *x* * *x* * *x* * *x* * *x* * *x

However, we can achieve the same final result with only four multiplications if we repeatedly take the square of each partial result, successively forming **(*x*^2, *x*^4, *x*^8, *x*^16).**

As another example, suppose we wish to calculate ***x*^11 mod *n*** for some integers *x* and *n*. Observe that ***x*^11 = *x*^1+2+8 = (*x*)(*x*^2)(*x*^8)**. In this case we compute ***x* mod *n*, *x*^2 mod *n*, *x*^4 mod *n*, and *x*^8 mod *n*** and then calculate **[(*x* mod *n*) × (*x*^2 mod *n*) × (*x*^8 mod *n*)] mod *n*.**

More generally, suppose we wish to find the value *a*^*b* with *a* and *m* positive integers. If we express *b* as a binary number $b_k\, b_{k-1} \cdots b_0$ then we have:

$$\mathbf{b}= \sum_{bi\#0} 2\wedge i$$

Therefore,

$$\mathbf{a\wedge b=a\wedge(} \sum_{bi\#0} 2\wedge i = \prod_{bi\#0} a\wedge(2i)$$

$$\mathbf{a\wedge b \bmod n= [} \prod_{bi\#0} a\wedge(2i) \mathbf{\,]mod}$$

$$\mathbf{n=(} \prod_{bi\#0} [a\wedge(2i) \mathbf{\bmod n]) mod\ n}$$

We can therefore develop the algorithm for computing *a*^*b* mod *n*, shown in Figure 1.

Note: that the variable *c* is not needed; it is included for explanatory purposes. The final value of *c* is the value of the exponent.

```
C ←0,f ←1
      for I← k downto 0
   do C ← 2 * C
          F ← (f*f) mod n
      if bi=1
            then C←C+1
                        f ← (f*a) mod n
return f
```

Note*:* The integer b is expressed as a binary number $b_k\, b_{k-1} \cdots b_0$

Figure 2: Algoristhm for Computing ***a*^*b* mod *n***

3.1.4 The Security of RSA

Four possible approaches to attacking the **RSA** algorithm are:

- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks:** These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

The defense against the brute-force approach is the same for **RSA** as for other cryptosystems, namely, to use a large key space. Thus, the larger the number of bits in *d*, the better.However,because the calculations involved, both in key generation and in encryption/ decryption, are complex, the larger the size of the key, the slower the system will run. In this subsection, we provide an overview of mathematical and timing attacks[1].

3.1.5 PROBLEM with RSA

We can identify three approaches to attacking **RSA** Mathematically.

1. Factor n into its two prime factors. This enables calculation of **φ (n) = (p - 1) ×(q - 1),** which in turn enables determination of **d = e^-1 (mod φ (n)).**
2. Determine **φ** (n) directly, without first determining p and q. Again, this enables determination of d = **e-1 (mod φ (n)).**
3. Determine d directly, without first determining **φ** (n)[1].

3.1.6 TIMING ATTACKS

If one needed yet another lesson about how difficult it is to assess the security of a cryptographic algorithm, the appearance of timing attacks provides

a stunning one. Paul Kocher, a cryptographic consultant, demonstrated that a snooper can determine a private key by keeping track of how long a computer takes to decipher messages .Timing attacks are applicable not just to **RSA**, but to other public-key cryptography systems. This attack is alarming for two reasons:

It comes from a completely unexpected direction, and it is a ciphertext-only attack.

Although the timing attack is a serious threat, there are simple countermeasures that can be used, including the following.

• **Constant exponentiation time:** Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.

• **Random delay:** Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack. Kocher points out that if defenders don't add enough noise, attackers could still succeed by collecting additional measurements to compensate for the random delays.

• **Blinding:** Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what ciphertext bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack.

RSA Data Security incorporates a blinding feature into some of its products.The private-key operation ***M = C^d mod n*** is implemented as follows:

1. Generate a secret random number *r* between 0 and *n* - 1.
2. Compute ***C = C(r^e) mod n,***where *e* is the public exponent.
3. Compute ***M = (C)^d mod n*** with the ordinary **RSA** implementation.
4. Compute ***M=M r^-1 mod n***. In this equation, *r*-1 is the multiplicative inverse of ***r mod n***; It can be demonstrated that this is the correct result by observing that ***r^ed mod n = r mod n.***

RSA Data Security reports a 2 to 10% performance penalty for blinding[1].

3.1.7 Chosen Ciphertext Attack and Optimal Asymmetric Encryption Padding

The basic RSA algorithm is vulnerable to a **chosen ciphertext attack** (CCA). CCA is defined as an attack in which the adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key. Thus, the adversary could select a plaintext, encrypt it with the target's public key, and then be able to get the plaintext back by having it decrypted with the private key. Clearly, this provides the adversary with no new information. Instead, the adversary exploits properties of RSA and selects blocks of data that, when processed using the target's private key, yield information needed for cryptanalysis [1].

3.1.8 The complexity of an algorithm

The central issue in assessing the resistance of an encryption algorithm to cryptanalysis is the amount of time that a given type of attack will take. Typically, one cannot be sure that one has found the most efficient attack algorithm. The most that one can say is that, for a particular algorithm, the level of effort for an attack is of a particular order of magnitude. One can then compare that order of magnitude to the speed of current or predicted processors to determine the level of security of a particular algorithm. A common measure of the efficiency of an algorithm is its time complexity.

We define the **time complexity** of an algorithm to be f(*n*) if, for all *n* and all inputs of length *n*, the execution of the algorithm takes at most f(*n*) steps. Thus, for a given size of input and a given processor speed, the time complexity is an upper bound on the execution time. There are several ambiguities here. First, the definition of a step is not precise. A step could be a single operation of a Turing machine, a single processor machine instruction, a single high-level language machine instruction, and so on. However, these various definitions of step should all be related by simple multiplicative constants. For very large values of *n*, these constants are not important. What is important is how fast the relative execution time is growing. For example, if we are concerned about whether to use 50-digit (*n* = 10^50) or 100-digit (*n* =10^100) keys for RSA, it is not necessary (or really possible) to know exactly how long it would take to break each size of key. Rather, we are interested in ballpark figures for level of effort and in knowing how much extra relative effort is required for the larger key size.

A second issue is that, generally speaking, we cannot pin down an exact formula for f(*n*).We can only approximate it. But again, we are primarily interested in the rate of change of f(*n*) as *n* becomes very large. There is a standard mathematical notation, known as the "big-O" notation, for characterizing the time complexity of algorithms that is useful in this context. The definition is as follows: **f(*n*) = O(g(*n*))** if and only if there exist two numbers *a* and *M* such that:

$$|f(n)| \leq A^*|g(n)|, N \geq M$$

An example helps clarify the use of this notation. Suppose we wish to evaluate a general polynomial of the form the following simple-minded algorithm is from:

```
algorithm P1;
n, i, j: integer; x, polyval: real;
a, S: array [0..100] of real;
begin
read(x, n);
for i:= 0 upto n do
begin
S[i] := 1; read(a[i]);
for j := 1 upto i do S[i] := x □ □S[i];
S[i] := a[i] □ □S[i]
end;
polyval := 0;
for i := 0 upto n do polyval := polyval + S[i];
write ('value at', x, 'is',polyval)
end.
```

In this algorithm, each sub expression is evaluated separately. Each S[*i*] requires (*i* + 1) multiplications: *i* multiplications to compute S[*i*] and one to multiply by a[*i*]. Computing all *n* terms requires:

$$\sum_{i=0}^{n}(i+1) = \frac{(n+2)(n+1)}{2}$$

multiplications. There are also (n + 1) additions, which we can ignore relative to the much larger number of multiplications. Thus, the time complexity of this algorithm is **f(*n*) = (*n* + 2)(*n* + 1)/2**.We now show that **f(*n*) = O(*n*^2)**. we want to show that for a = 1 and M = 4 the relationship holds for **g(*n*) = *n*^2**. We do this by induction on n. The relationship holds for n = 4 because **(4 + 2)(4 + 1)/2 = 15 < 4^2 = 16.** Now assume that it holds for all values of n up to k [i.e**.,(*k* + 2)(*k* + 1)/2 < *k*^2].**Then, with ***n* = *k* + 1**,

Therefore, the result is true for ***n* = *k* + 1**.[1]

$$\frac{(n+2)(n+1)}{2} = \frac{(k+3)(k+2)}{2}$$

$$\frac{(k+2)(k+1)}{2} \text{ +k+2} \leq \text{k^2+k+2}$$

$$\leq \text{k^2+2k+1=(k+1)^2=n^2}$$

Therefore, the result is true for ***n* = *k* + 1**.[1].

IV. CONCLUSION

In this paper, we provide a survey of IDS technics and kinds of attack can permeate to computer networks. For this we find the solution with RSA Cryptography Algorithm that decrease intrusion detection system vulnerabilities .intrusion Detection Systems (IDS) have long had a problem with packet fragmentation. This was true five years ago and it is still a problem today. For years the IDS has suffered from several key ailments. Fragrouter will be used as our main packet fragmentation tool for this article, but we will also look at Metasploit Framework's built in fragmentation abilities as well. These fragmented attacks will be tested against the Snort IDS. Hackers can Permeate to IDS with fragmentation. This paper removes IDS weaknesses by RSA cryptography algorithm.

REFERENCES

[1] William Stallings ,cryptography and network Security ,Fifth adition 2011
[2] Seyed hasan mortazavi: data mining for Intrusion detection system, international conference On computer science and engineering-April 28th, 2012 – Vizag - ISBN: 978-93-81693-57-5
[3] Páez, R. Satizábal C., Forné J. Cooperative Itinerant Agents (CIA): Security Scheme for Intrusion Detection Systems, *Proceedings of the International Conference on Internet Surveillance and Protection* (ICISP). ISBN:0- 7695-2649-7. Pag. 26-32. 2006.
[4] RSA Security. The RSA security survey of San Francisco. RSA Security Inc. http://www.securitymanagement.com/
[5] Tian Fu and Te-Shun Chou International Journal of Computer Engineering Science (IJCES) Volume 2 Issue 5 (May 2012) ISSN : 2250:3439