

6.805/STS085: The Threats of Distributed Cracking:

Andrew Twyman

Paper for MIT 6.805/STS085: Ethics and Law on the Electronic Frontier, Fall 1997

Contents

- Introduction
- Distributed Cracks: Past and Present
 - The SSL Challenge
 - The RSA Secret-Key Challenges
 - Other Distributed Contests and Efforts
 - Future Potential
- Misuse Potential
 - Public Distributed Efforts
 - Local Networks
 - Remote Hacking
 - Web-based Computing
 - Level of Potential
- Vulnerabilities
 - Threat Sources
 - Types of Vulnerabilities
 - Single-Key Vulnerabilities
- Solutions
 - Increased System Security
 - Increased Key Security
- Conclusions
- Sources

Introduction

The encryption challenges of past years have provided an opportunity for the demonstration both of the potential and of the risks of today's computer networks. Most of the successful cracks of these challenges used distributed methods, dividing the work of a brute-force search for a key among tens, hundreds, or thousands of individual computers by way of a network. In doing so they simultaneously demonstrated the shortcomings of today's encryption systems, and the potential of distributed computing to solve computationally intensive problems. The implications of these efforts for distributed computing, though, are not all good, as they present the very real possibility that similar methods could be used to crack encryption keys for illegitimate purposes. This is an important threat

which must be addressed and dealt with if encryption systems are to remain secure and if distributed computing is to enter wider use in the future.

To demonstrate this, we first explore the history of encryption challenges and the use of distributed computing to crack them, and combine that information with trends in computers to estimate the future potential of distributed computing. We then explore situations and scenarios in which distributed computing methods could be misused in order to illegitimately crack encryption keys, demonstrating that the threat of distributed cracks is not an empty one. Finally, we present key vulnerabilities in modern encryption systems which are most susceptible to attack using distributed cracking methods, and which allow even the small number of cracks possible with such methods to be put to devastating use. The threats presented by distributed computing are not impossible to avoid, and we also present ways to reduce or eliminate those threats, along with the difficulties inherent in doing so. The matter thus illuminated is one which deserves serious thought as computing networks develop, and which must be satisfactorily resolved if those networks are to take their prominent place in the daily business of society.

Distributed Cracks: Past and Present

Although parallel and distributed computing is certainly not a new concept, the potential to link thousands of end-user machines to solve a single problem did not truly exist until recent years. The achievement of a critical mass of users on the Internet, as well as the existence of an issue such as encryption which was both important (or at least of interest) to many Internet users and suitable to distributed computing made the current growth of distributed efforts possible. Cracking of encryption by brute force methods is a trivially parallelizable computation problem, and thus lends itself very easily to this method of attack. What was also needed, though, was an incentive to put that potential to use. In the famous distributed efforts of recent years, that incentive has come in the form of public contests or challenges, usually with cash prizes. One of the first demonstrations of the potential of shared spare cycles was in response to the 1991 RSA factoring challenge, a contest which called upon participants to factor a composite of large primes. There have been other efforts carried out independent of stated contests or challenges, but it is the encryption-related challenges which have garnered the most public attention.

The SSL Challenge

The SSL Challenge of 1995 gained a great deal of attention because its results were directly relevant to the quickly-growing Worldwide Web. The Challenge asked participants to crack a web session encoded with Netscape's SSL encryption, based on the RC4 stream-cipher with a 40-bit key. The encryption cracked in the challenge represented the default encryption available for secure web transactions, at the maximal level of strength then allowable under government export limitations. To drive home the security risk at hand, the contest session was a fictional credit-card transaction. Though a widely distributed effort was in the works, the winning crack came from a different source, or rather from two. The crack was achieved by two efforts ending within 2 hours of each other. Both took approximately 8 days to locate the key, and both used a combination of a locally distributed system (over networks of workstations in an academic setting) and computation on a small number of massively parallel machines.

Though these cracks had their desired effect of demonstrating shortcomings in available encryption standards (whether they lead to a solution still remains to be seen), they also demonstrated the potential of distributed techniques. What is notable is that they did so not on the huge scale of the later RSA challenges, but on a smaller scale. Both efforts used slightly more than 100 workstations, a number quite feasibly available to a user or sysadmin on an academic or corporate network, to crack an encryption system which even today is used to protect credit card numbers and other sensitive data. When later that year the crack results (including time) were repeated on a single \$100,000 parallel rendering workstation, it was a further demonstration of the potential of small scale efforts, as well as the rapid increase in power that always exists in computing technology.

The RSA Secret-Key Challenges

In January of 1997, RSA Data Security announced a series of cryptographic contests. Their goal was to "quantify the security offered by the government-endorsed data encryption standard (DES) and other secret-key ciphers with keys of various sizes." [1] There were thirteen challenges, with prizes ranging from \$1,000 to \$10,000. All of the contest messages consisted of some known text ("The unknown message is: ") followed by the unknown message, thus allowing a simple known plaintext-ciphertext pair attack. The first of the challenges used the 56-bit DES block-cipher, and the others used the RC5 block-cipher at varying key lengths. Given the large key size of most of the contests, large-scale distributed efforts were probably the only feasible means of meeting the challenges. Any other computer of the necessary speed and power would be too expensive to dedicate to the task for the vast amounts of time necessary to crack the RSA challenges, and the smaller-scale efforts that had cracked the SSL challenge would take far too long to reach the answer.

The efforts which have grown in response to the RSA Secret-Key challenges were more sophisticated than the thrown-together responses to the SSL challenge, and continue to become more so. The DES Challenge was the first to fall, cracked by Rocke Verser's DESCHALL [3] effort, one of several distributed efforts to make the attempt. There were also several distinct efforts to crack the RC5 challenges at key lengths up to 56-bits, but with the move on to 64-bits the focus has been on Distributed.Net [4], a nonprofit organization which has grown out of the challenges. With its numerous key servers, and thousands of participants (many acting as members of organized teams) Distributed.Net achieved a maximal key search rate of approximately 7.2 billion keys per second during their work on the 56-bit RC5 challenge, which they won. The client software for the Distributed.Net efforts is freely available in versions optimized for most available PC and workstation platforms, though source code and detailed design information is not available. Their membership continues to grow, and their efforts continue, currently at a key rate of 17.2 billion keys per second directed toward the 64-bit RC5 challenge and the DES Challenge II mentioned below.

Another RC5 effort deserving of mention is the RC5 Java Breaking Effort, located at MIT. Though it is far smaller than the Distributed.Net effort it is notable for the method by which it allows users to participate. Rather than distributing specialized clients for each possible workstation type, the effort uses a Java applet to perform the key search. Though this method does suffer from the slowdowns and overhead inherent in Java, it is also completely platform independent, allowing a single piece of contest code to run on any client machine. Just as importantly it makes it trivially easy for users to join the effort. They need only click on a link to access the applet page, and then the applet will run

the challenge code until it is terminated. This type of universality and ease of participation is sure to be a boon to distributed efforts in future, but it can also present its own risks as will be discussed later.

Other Distributed Contests and Efforts

Though the RC5 Challenges have received the most press of late, there are other problems being solved by distributed efforts. Many such efforts are based on difficult mathematical problems, being solved primarily for academic interest rather than for cash prizes or political statements. The search for large prime numbers and factoring of large composite numbers are favorites at Marsenne.Org[5], but they are joined by other more unusual problems. The Marsenne.Org efforts are human-coordinated, with participants simply claiming ranges of numbers to test and then sharing their results, but the problems could easily be brought into an automatic coordination framework like that of Distributed.Net. On less mathematical subjects, other distributed computing efforts include a SETI project to scan radio data for patterns which could indicate signs of extraterrestrial life, and a current effort to develop a distributed chess engine.

Very recently (on January 13, 1998) RSA issued a second DES challenge, which only gives prizes for cracking efforts which beat the previous best time by at least 25%, with the prize amount increasing based on the improvement. Distributed.Net has created their first dual-purpose clients to allow simultaneous work on the DES Challenge II and the 64-bit RC5 challenge. Users choose which challenge to work toward (though Distributed.Net has stated DES as its current priority), and the client will automatically switch to the other when the first one ends.

Future Potential

The dual-purpose DES/RC5 clients are only the first step in Distributed.Net's future plans. Beyond their sheer statistics, the interesting thing about the Distributed.Net effort is that their long-term goal and vision goes beyond encryption challenges into the realm of general-purpose distributed computing. Their v3 clients which are currently under development will take the form of a general-purpose program framework for running and coordinating distributed computations. The client will then be able to take any of a number of modular computation cores created to solve specific problems, allowing a single framework of participants and servers to solve many different problems, either simultaneously or in sequence, as decided either by individual users or by a central decision-making body.

Distributed.Net's plans look toward what may well be a very significant future for distributed computing. The increasing popularity of the Internet, and of computers in general has been unmistakable in recent years. The number of personal computers is growing at a phenomenal rate, as is the number of those computers which are connected to some sort of local network, and usually at least indirectly to the Internet. An additional factor is the growth of the computational ability of modern computer technology, subject to the rule of thumb of doubling of power approximately every eighteen months. With the combination of these factors it is clear that the sheer amount of computation power available in the world is rapidly increasing. As it has been since the decline in the popularity of mainframe systems, this computation power is largely scattered among individual

workstations, but more and more of those workstations are now being connected together. The amount of computation power which goes unused in such a situation, taking into account the many PCs left idle on office desks every night, is truly phenomenal, and thus there is clearly potential for efforts to make use of that extra potential by putting lost cycles to use for a distributed computation.

It is quite feasible, therefore, that the future could see widespread use of distributed computing to reclaim unused time on workstations. In one of the most likely scenarios, corporations which have need of large computations could harness the desktop computers of their employees rather than spending more on specialized parallel machines or supercomputers. Specialized "farms" of rendering workstations are already used in the production of high-quality graphics, and as the graphics capabilities of the average workstation increase it becomes more and more feasible to replace or augment that function with distributed use of idle cycles. On a larger scale, it is quite possible that more public efforts, such as government sponsored research projects, could be both public enough and popular enough to gather thousands of participants on the Internet for distributed efforts, as Distributed.Net has done for the RSA challenges. The use of contests and prizes as incentive may well continue, or there could be a move to a more general method of incentive for the donation of processing power, perhaps organized by Internet service providers in terms of discounted usage fees (sort of an Internet equivalent of frequent-flyer-miles). In the most general case, it is quite possible that at some point in the future all, or at least many of the world's personal workstations will be part of one or several massive general-purpose "distributed computers" which could then be used to solve computationally intensive problems which would now be considered completely infeasible. The possibilities, if not limitless, are at least vast and impressive.

Misuse Potential

Clearly, distributed computing has great potential for legitimate and productive use of spare computing power in the solution of difficult problems. In the realm of distributed cracks specifically, the challenges and distributed efforts to date have provided useful demonstrations of the weaknesses of current encryption standards, as well as acting to increase public awareness of the issue of encryption and security. Like most powerful tools, though, distributed computing is not without its potential for misuse, and the field of distributed cracking is one of those with the greatest potential. The legitimate uses for cracking are far less numerous than the illegitimate uses, and in the past it has been the unavailability of techniques and equipment necessary, as well as the lack of widespread use of encryption outside of government circles, that has kept those illegitimate uses from becoming widespread. The increasing use of encryption for sensitive business and personal data provide the incentive for the illegitimate cracking of encryption keys, and distributed computing methods have the potential to provide the means. Though most distributed efforts so far have been very public and directed toward legitimate ends, there are many ways in which the spare cycles of personal workstations could be harnessed to more illicit ends. If distributed computing becomes truly ubiquitous in the future that risk may be multiplied greatly, but even today there are clear potential risks.

Public Distributed Efforts

The simplest situation to imagine would grow directly out of current distributed cracking efforts.

Distributed.Net and other efforts already have a wide base of participants, and the publicity brought about by encryption challenges have helped that base to grow truly enormous. The clients distributed by those efforts are already highly specialized to cracking of some of the best contemporary encryption standards, DES and RC5. How difficult would it be, then, for an organizer or programmer in one of these efforts to modify the client or server software slightly so that the "fastest computer on Earth"[4] was dedicated not to cracking RSA challenges, but instead to cracking credit-card transactions, corporate data, or government secrets? The answer is that it would not be very difficult at all. Only the plaintext-ciphertext pair would need to be replaced, and the thousands of participants would obviously go about aiding in wire fraud. Given the high level of luck involved in brute-force searches (since the real key can be anywhere from the very beginning of the search space to the final key tested), it is unlikely that anyone would notice the change in the net results if even a large fraction of a distributed effort's computation was dedicated to a purpose other than its stated one. As the generality of the software used in such efforts increases, so does the risk of such misuse. With the current Distributed.Net clients, it would already be trivial to make modifications such that the servers could switch the client software between DES and RC5 cracking, overriding user preferences. With their next generation of modular clients, it may well be possible to download any computation core without the user's knowledge, dedicating the entire effort to whatever task the organizers choose, and this is only a fraction of what might be possible if distributed computing ever became truly universal, with every workstation connected as part of a general-purpose distributed computer.

Of course, all of this is only speculation. All of these misuses are possible, but there is no reason to suggest that they are actually occurring. For the most part, the priorities and interests (not to mention morals) of those organizing the distributed efforts keep such things from happening, as such a plan could only work for as long as all of those involved kept it secret. The other guard against such misuses is the very public nature of distributed cracking efforts. All that is required to put an end to misuse of such a public effort is for a single user to take the time to analyze the code or the network traffic of a client program. In situations like that at Distributed.Net where the source code and network protocols are not published such a task is somewhat more difficult, but there are enough resourceful, curious, dedicated, or bored people in the world of computers that someone would be likely to discover a misuse. If such a thing were to happen, then it would be sure to garner great public attention. The distributed effort in question would be likely to be shut down very quickly, and users would become more hesitant to join such distributed efforts in future. For the sake of the good that distributed efforts have done for the encryption field, then, it is best to hope that such a misuse never takes place. However, the potential does exist, and it may well just be a matter of time before someone is either resourceful or lucky enough to attempt it and succeed, at least for a short time or on a smaller scale.

Local Networks

Though on a smaller scale than the massive efforts that have cracked DES and RC5 challenges, the cracks of the SSL challenge demonstrated that distributed methods could have meaningful results when applied to smaller networks, on the order of 100 workstations. This is a realm where misuse would be far easier to accomplish than in the massive public efforts. In an environment such as a university or corporation where direct access to workstations is possible, especially during off hours when users are absent or sparse, it would be easy for a single user to put those workstations to use in distributed computing. Athena at MIT has already seen many demonstrations of this, ranging from

computer graphics students hogging workstations for rendering projects, to students arranging for public workstations to run background processes to participate in the public DES or RC5 challenges. Simply dedicating machines to a distributed task during off hours is the simplest method, but it only takes someone with a small amount of programming and security expertise, especially with sysadmin privileges, to put workstations to work far more effectively. It is relatively easy to create a program which runs in a way that will be nearly invisible to the average user, resistant to simple attempts to remove it, capable of automatically restarting itself if canceled, and intelligent enough to slow its operation so as not to cause noticeable performance degradation when a workstation is in use.

Such techniques have quite foreseeable legitimate uses, such as dedicating a corporate network to some business-related computation in a way which is nonintrusive to users, but the potential and incentive for misuse is far greater. In addition, the safeguards which help to prevent misuse of public distributed efforts do not exist in this situation. Given a few hacking skills, it is relatively easy for a user to set up such theft of cycles in a way which would be difficult if not impossible to trace back to him. This is especially true on many academic networks, and even some corporate networks, where internal security is minimal. If even MIT, where security may not be at its tightest but is at least well designed and implemented by knowledgeable people, is susceptible to such things, then there can be little hope of completely avoiding such misuse of computing at institutions with fewer resources in money, manpower, and knowledge to dedicate to the security of their computing facilities.

Remote Hacking

Though direct access to a local network makes it far easier to put the workstations on that network to use for illegitimate distributed cracking, it is by no means impossible for a hacker to do the same thing without that local access. In fact, the prospect is far more frightening, because it allows the possible scale of computation power gained to increase far beyond the number of machines that a single user could access, and back into the same category as the large public efforts like Distributed Net. In 1988 the Internet received a frightening demonstration of the capabilities of an automated program to take control of machines on a vast scale, in the form of the Internet worm. The worm spread itself from machine to machine automatically, taking advantage of holes in the security of UNIX operating systems, and was very difficult to eliminate because any "cured" machine still connected to the network could be quickly reinfected. The worm was intended to be harmless, but a bug caused it to infect single systems multiple times, causing those systems to eventually collapse under the weight of too much computation and network load.

Though the security holes used by the Internet worm have largely been patched, new holes are constantly discovered in today's electronic security systems, and patches for those holes always lag behind their discovery, especially at less maintained or lower security sites which are slow to respond. Thus it is still quite possible that a program like the worm could again be released on the Internet and be as difficult to stop and trace as the original. Indeed, it could be far more difficult. While the bug in the original worm was the cause of its massive havoc, it was also the reason that the worm was discovered so quickly and so much time and effort was dedicated its eradication. Assuming a programmer who valued subtlety over destruction, a neo-worm could easily spread unchecked, and undiscovered for a long while. With the growth of the Internet in the past ten years, the number of machines possibly infected could be phenomenal, especially if the neo-worm could infect non-UNIX operating systems such as Windows 95.

Once that access is gained, the uses to which it could be put range from the harmless technical demonstration which was the purpose of the original worm, to massive destruction of data. Somewhere in between, though, lies a quite solid possibility that such a neo-worm could be put to use not to steal or destroy data, but to steal cycles. Once infected, a computer could use its idle cycles to contribute to a distributed computation, while the worm remains safely hidden from view, carefully rationing its CPU usage so as to remain unnoticed, and modifying system software to foil detection efforts. The only major limitation of such a use would be the need for a central server to coordinate the computation, certainly a major technical problem in the general case, if the server is not to be revealed as soon as a single copy of the worm is discovered. In the case of such an easily parallelizable problem as brute-force cracking, though, it is likely to be a solvable one. In the simplest case, the server can itself exist on a hacked machine, and all the hacker need do is occasionally check that machine for found keys. If the server is found and shut down, the hacker can simply use periodic data on which keys the server had searched to set up another server to search the rest of the key space, with minimal loss of effort. More elegant solutions could hide the server behind multiple levels of redirection, or perform purely through communication between infected machines, contacting a server only in the event that the key is found. In fact, the infected machines could just as easily be instructed to search the key space in a random order, thus completely eliminating the need for coordination, while still (probabilistically) allowing the key to be found without too much delay. The RC5 Java Breaking Effort, in fact, uses this randomized search approach to save in coordination hassles, so there is no reason that a hacker could not do the same.

Web-based Computing

Relating very directly to the potential of distributed computing is the development of computer languages and protocols explicitly designed to allow code from a server to be run by a client's machine. These technologies, most notably Java and ActiveX, also have a great potential for abuse. Security and the danger of virus or worm-like programs has always been a major concern with these technologies, and it has been addressed with varying levels of effectiveness. Java's security, though not perfect, is relatively tight. Downloaded programs are run through an interpreter which can act as a watchdog, ensuring that they do not access areas of the computer that are off-limits, such as the operating system or drives. That watchdog also controls which servers a program can communicate with through the network, usually limiting it to the machine from which it was downloaded. ActiveX has nothing approaching that level of security. The code downloaded is native machine language, and is run in an environment with no watchdog, in which it can quite literally take complete control of the user's machine (quite a frightening concept considering that the operating system which supports ActiveX is the most widespread of them all). Despite their differing levels of security, neither of these protocols puts any limitation on the type of computation which can be performed, quite simply because it was not thought to be a problem. Depending on the system you use and the security holes that exist, the cute animation program on your favorite web page may not be able to reformat your hard drive, but there is nothing at all stopping it from running a brute-force cracking program on the side while it dances across your screen, and it is likely that there will be no indication of its activities other than a higher load on your CPU.

The extent to which such techniques can be put to use for distributed cracking is variable, but the

potential certainly exists. In either system an applet or control can easily be hidden in a web page even if it makes no display to the screen such as an animation, and only a look at the page's HTML code is a surefire method of finding such a thing. Such a program can also, in both systems, be made to continue running after the user leaves the web page in question, for at least as long as the user runs their browser, which is often quite a long time, considering the number of people who simply leave a browser window open at all times. In Java, that is the extent of the theft of cycles which is possible. When the browser closes, so does the Java interpreter and any programs that it was running at closing time. ActiveX, on the other hand, allows far more. Once an ActiveX control is running it can gain access to the user's entire system, perhaps spawning off a new background process which will run long after the browser is closed, or even modifying the user's system files so that this process is restarted at boot-time.

The difficulties in preventing these misuses are also variable. In Java, a dedicated and knowledgeable user could easily run a monitor on the CPU usage of his Java system, and terminate the offending program if it is noticed to be taking up a large percentage of computing cycles. Unfortunately, CPU usage is the only useful statistic which can be monitored in this case. Actually having the Java watchdog detect what computation an applet is performing is impossible (literally an uncomputable problem, equivalent to the halting problem), and with only CPU usage as a benchmark it is hard to distinguish between a distributed cracker and, for instance, a CPU-intensive MPEG renderer. In ActiveX a similar precaution would be possible, but since it would have to use the native operating system's monitoring facilities, and since the offending ActiveX control could have direct access to those facilities, it would be easier for it to "cloak" itself to avoid detection, using the same methods that would be used to hide a process spread by direct machine access or worm-like hacking techniques. The other major difficulty in using Java or ActiveX to steal computation power is in distributing the code in the first place, since the user must first access a web page containing the code in question, and indeed in Java they must access that page each time they run their browser in order for the code to be run. With the web-surfing tendencies of today's Internet users, putting a piece of code somewhere that it will be accessed is not particularly difficult. Having it accessed often by the same user so as to bypass Java security might be more difficult, but the power that could be wielded through illegitimate use of a popular web site, such as a search engine, in this case is great.

Of course, the web page, and its hosting server, act as a vulnerable point here even more than in the worm-like hacking case. Once a single user notices the offending code, be it through Java monitoring, CPU slowdowns, or taking a look at web page source-code, they can arrange to have the site closed down, especially if it is not under the direct control of the cracker in question. Of course, the effectiveness of such methods in stopping a dedicated cracker may be limited. Similar methods are already used to attempt to stop the posters of unwanted junk-email, by contacting the sysadmins of their service providers, and often either the sysadmin doesn't care or, more often, the spammer simply finds a new account from which to ply his trade. It is quite possible that the theft of computation could follow a similar pattern. Presumably, since encryption cracking is closer to actual crime than unwanted email the effort taken to stop it would be greater, but the potential still exists for a cracker to elude detection and continue his efforts indefinitely.

Level of Potential

These methods are all feasible ways in which a knowledgeable or dedicated cracker could gain the

use of large numbers of computers for distributed cracking efforts without the permission of the users of those computers. Of course, all are speculative scenarios, and the challenges, both technical and practical, involved in carrying out any of them are significant. Still, the same is true for attempts to prevent or stop such a cracker's efforts. Thus, while panic is perhaps not the proper response, complacency probably isn't either. There is a real possibility of the misappropriation of computing resources in order to perform a distributed attack on an encryption system. As the use of computers and the Internet grows, along with the use of encryption, that potential will continue to increase. If distributed computing truly becomes ubiquitous in the future, then that potential may well become one of the primary issues facing the designers of the computers of tomorrow.

Vulnerabilities

Given the potential that has been demonstrated for encryption cracking using distributed computing techniques, as well as the potential for the misuse of these techniques, how much of a threat do these techniques really present to the use of encryption for secure communications on the Internet and in other electronic communications? The threat is a real one, but analyzing it is by no means easy. An important thing to remember is that the scale of the threat, in terms of the number of keys crackable and in how long, is probably relatively small. It took a combination of 100 workstations and a few parallel machines 8 days to crack 40-bit encryption in the SSL challenge. The DESCHALL distributed effort took 140 days to crack 56-bit DES encryption, and the deadline for the prize in the second DES Challenge is being set at 68 days (a deadline that Distributed.Net hopes to meet). The time taken to crack larger keys would be correspondingly larger, and each such crack results in only a single key. In this case, the cost per key cracked is usually the number used to analyze the level of threat, in terms of whether any attack is worthwhile for the cracker. After the SSL challenge Netscape estimated the cost per 40-bit SSL crack at \$10,000. The single-machine crack which followed lowered that estimate to little more than \$500. However, with distributed efforts like DESCHALL and Distributed.Net, coming to a single figure is a difficult proposition if possible at all. In a hypothetical situation where processing power on many machines was being stolen by a cracker, then the cost of the workstations involved would be irrelevant, and the cost to the cracker himself could only be reckoned in terms of the time, effort, and risk involved.

Even if no analysis as simple as a single dollar value can be performed, it is clear that the effort and risk involved in using distributed techniques to crack encryption are large, while the sheer number of keys which can be cracked is relatively small. There are, of course, factors which affect even this qualitative estimate. For instance, the brute-force cracking used by all public cracking efforts is a very random process, with the possibility of finding a key in the first few seconds of the search, or at the very end of the search space, or anywhere in between. In addition, there exist cryptanalytic methods of cracking, especially in situations where single keys are used for multiple messages or sessions, which provide improvements on the search time of brute-force techniques. Bugs in encryption algorithms or their use (such as the infamous, now fixed, Netscape SSL random-number flaw which reduced the key search space to about 1000 keys) can make an intelligent cracker's job even easier. Even with this qualitative knowledge of the level of the threat it is possible to pinpoint the greatest vulnerabilities to that threat, as well as who is most likely to benefit from taking advantage of those vulnerabilities.

Threat Sources

The general tool of encryption cracking could be applied by many different people, in many different ways, and on many different scales. Scale is truly an important factor, since the total key-cracking capability of any distributed cracking method, no matter how wide its support, could probably only barely (if that) rival the capabilities of a dedicated agency such as the NSA, with extensive supercomputing resources and the monetary resources to dedicate time, man-hours, and specially-designed hardware and software to the task. However, intelligence organizations anywhere close to rivaling the NSA are very rare in today's world, and their numbers and actions tend to be limited by their very scale and cost, and by the policies of the governments which support them.

Even if it cannot rival such resources, distributed cracking can quite feasibly put significant capabilities into the hands of far smaller groups of people with far fewer limitations. Foreign intelligence operations are probably not the most likely use of distributed cracking, though they are a possibility. Where it is likely to be seen is in organized crime, terrorism, white-collar crime, small-time criminal activity, and individual actions by hackers. In its largest scale, what makes distributed cracking a threat is its ability to bring smaller, less legitimate organizations into the same realm as government agencies, at least for a small number of cracks. There are probably more than enough criminal or terrorist organizations, or simply resourceful individuals who would be more than willing to reap some benefits, turn some profit, or do some damage by using distributed cracking techniques, and who can certainly afford to pay the few computer-savvy individuals it would take to do so. On the smallest scale, even on the level of the use of workstations on a corporate or academic network to crack a few keys, distributed cracking can still present a threat to corporate security or personal privacy. An individual reading another individual's email or data is still an invasion of privacy and possibly a crime, even if the FBI and NSA is unlikely to waste resources on stopping it. They are also far more likely to become interested if the data in question contains corporate trade secrets.

Types of Vulnerabilities

Most of the encryption challenges to date have applied directly to encryption algorithms, but their implications go well beyond those algorithms, as there are a multitude of places in which those algorithms, or other algorithms susceptible to brute-force cracking are used. More traditional security means such as passwords, pin numbers, and credit card numbers can easily be found by brute-force search methods, and the search space involved is usually far smaller than that of the more secure encryption keys. Within the realm of direct use of encryption algorithms, encrypted email, while an obvious example, is probably one of the least major threats due to the level of sensitivity of its typical content. Encryption of online sessions is probably a far more tempting target, and was the example used by the SSL challenge. Simple web transactions are already containing credit card numbers regularly, and as the popularity of commerce on the web increases so will the percentage of such transactions containing such information. Of course, session-encryption tends to use a new key for each session, established for the new session by key-sharing techniques and then thrown away, meaning that a long-running distributed crack would allow access only to that session's data. Digital cash, which is just beginning to catch on in online commerce also has little alternative to using encryption techniques for authentication. If poorly designed, digicash systems could well provide ripe targets for crackers using distributed methods, but a well-designed digicash system will also use one-time keys.

The most tempting targets are not the one-time session keys, but keys which are reused. Such a key once cracked could be reused indefinitely, until its legitimate user changes it or discovers its misuse. Many authentication systems rely on encryption keys which change only rarely. For instance, if a public key must be registered with a certificate authority, and distributed to all of those who need to communicate with the holder of the private key, then the user has good incentive to avoid repeating that hassle by changing keys too often. If encryption is to be built into automated systems, especially those which are hardware rather than software based, then there is an even greater likelihood of keys remaining unchanged. In the original Clipper proposal, for instance, a Clipper phone had a single dedicated key hard-coded into its Clipper chip. While the chip itself was supposed to be tamper resistant to avoid recovery of the key, a brute-force crack of the key would allow a cracker to listen in on telephone conversations indefinitely, until the phone in question, and possibly many others had been replaced.

Single-Key Vulnerabilities

Logistical limitations make it inevitable that keys will be reused, often for far too long. They also tend to lead to systems where a single key or set of keys can provide great dangers (or benefits, depending on one's point of view) if cracked, by providing access either to other keys or to particularly sensitive data or computer systems. It is these keys which provide the most likely, and the most frightening potential targets for crackers, and which provide the greatest threat for the use of distributed computing methods. Cracking such a key is quite likely to be worth any effort or risk that a cracker undergoes in the process. In a simple example, consider the cracking of the session key that protects a secure telnet connection, say one using SSH or Kerberized telnet. The actual contents of the session, commands or maybe a few email messages, is probably not particularly sensitive, but the session will probably also contain the password of the user logging in, which the cracker can steal and use. If that password belongs to a sysadmin or other user with great privilege, then it could give the cracker access to a great amount of data or resources. If it led to a corporate system, then it could allow theft or destruction of corporate data (trade secrets, perhaps) worth a great deal. If it led to a bank system, it could allow theft of a great deal of funds. If it led to a critical infrastructure system, such as air-traffic-control, power-plant control, or emergency communications, then the password could be of great use to a terrorist, foreign agent, or random whacko wishing to cause a great deal of damage. Similar scenarios can be conceived involving other access codes, such as credit card numbers, bank pin numbers, etc.

Compounding this situation is the ability of some passwords or keys to be used to access others. A sysadmin's password could be used to gain access to the accounts of any users on that machine. Those accounts might contain copies of encryption keys on disk. Sysadmin access could also allow a cracker to listen in on the sessions of those users, gaining the passwords for other systems they log in on, or the keys they use in other encrypted communications. The worst possible example of this comes in the phenomenon of key escrow. The escrow system in question could be one of the possible systems proposed by the government for the accessing of encrypted communications by law enforcement, or it could be a corporate escrow system used to protect the corporation from data loss. In either case, a cracker gaining access to the escrow server or master keys by whatever means could then gain access to the keys of all users registered with that escrow server.

Any other major centralized encryption system would have similar weaknesses, since their logistics necessitate central and powerful keys which cannot be changed often. For instance, some of the proposed key escrow systems include a "government key" which would be used to encrypt a copy of the session key with each encrypted session. While allowing easy law-enforcement access to encrypted communications, that single key, if cracked, could also allow anyone else that same access. Outside of the field of key escrow, it is widely agreed that a widely useable encryption system will require some sort of key certification authorities to ensure the genuine nature of public keys by signing them. This allows users who trust the authority to then be able to trust the identities of any certified parties they communicate with, thus allowing business to be carried out over the network. If the signature keys of the authority could be cracked, however, then the cracker could impersonate any individual who uses that certificate authority, and the possibilities of such capability, at least in an environment where online transactions and business are commonplace and trusted, are limitless. These are tempting vulnerabilities indeed, and difficult ones to secure, both technically and logistically.

Solutions

The risks presented by distributed cracking methods are all too real. Even if none of the scenarios outlined here ever come to pass, the risk will always exist, and may well increase. The Internet, and electronic communications in general are poised to take on a strong and valuable role in the business and life of the future, but they cannot take that role if they are not trusted. Many people, myself included, are still hesitant even to send their credit card information through an encrypted web connection due to the risks which exist, and the plethora of password sniffers and other such threats highlight the risks of today's largely unencrypted systems. If tomorrow's security systems are to be trusted to allow business as usual to occur electronically, then they must first deal with the threats to their security. No perfect solutions can exist. Any security system has its flaws, and this is just as true in computers as it is in real life. Additionally, individuals or organizations which wish to break electronic security systems will always have Robert Morris' three Bs (Burglary, Bribery, and Bugged equipment - often far more useful tools than cryptanalysis) to fall back upon. Good solutions do exist, though, and a good solution is necessary to build the confidence that future users of the network infrastructure must have in order for it to be the benefit that it has the potential to be.

Increased System Security

Changes and tightenings of the security systems of the Internet, both in software used and in the human and computer infrastructures which exist to promote and ensure security, apply to the topic of distributed cracking in two major ways. The most obvious is to make it more difficult for distributed methods to be misused in the first place. Many of the misuse scenarios outlined depended in flaws in existing security systems, either electronic or human, to allow the cracker to gain access to the distributed resources that he used. Patching of remote access flaws, tighter limitations on the capabilities of programs downloaded from the network, and increased user awareness and control of the activities of his computer (even when he isn't present) will all go a long way toward reducing the possibility of those scenarios taking place, or at least raising the hurdles which must be passed to put such a scenario into effect. Certainly if distributed computing does enter widespread use, then care will need to be taken to ensure that those resources are properly used. Perhaps the notion of

certification authorities could be applied to distributed computing to allow users to trust the servers that to which they dedicate their cycles, but the technical problems of that or any other method of trustable distributed computing are far from solved, or even simple. Increased security can also apply to the systems which are attacked by distributed methods. If there are fewer cases of single keys with significant power, or if keys are changed or discarded more often, then each of distributed cracking's limited number of cracks becomes less useful.

Of course, such security when taken to extremes can become as much of a detriment to the network and its users as the threats of crackers would be. In order to be useful, computer networks must not only be trusted, but they must also be easy to use and effective. If users and systems become bogged down in security measures, or if security unnecessarily limits what a user can access or do on the network, then use of that network will not grow as it should, and the usefulness of that network will be limited. Even distributed computing itself has great potential for usefulness, which should not be stunted by the paranoia of security experts worrying about too many threats. Finding the happy medium in this situation is no easy matter, especially as the size and complexity of the systems and infrastructures under consideration grows. Great effort will need to be dedicated to ensuring that those systems maintain an acceptable level of security, while not sacrificing their usefulness.

Increased Key Security

The other obvious solution, and one which is far less complex or difficult to implement, is simply to make the individual keys used in the security and encryption systems of the world less susceptible to distributed cracking attacks. The simple way to do so is through increased key length. The nature of encryption keys is that the difficulty of brute-force cracking attacks increases exponentially with increased key length, and thus it is quite possible to use encryption keys which could not be cracked in a reasonable period of time even by all of the computers in the world working in concert. Most modern crypto systems can trivially be modified to use keys which are not merely difficult, but near-impossible to break. Use of such keys, at least in an encryption system which ensured that brute-force was the best possible attack, would make any tricks used by distributed crackers irrelevant.

There are two limitations currently placed on key length. The first is technical, simply that algorithms that use larger keys, while not necessarily more complicated conceptually, can often become more computationally-intensive, or in the case of hardware implementations require more physical resources to implement. Thus, for the purposes of performance and affordability, encryption suppliers and users have tended to choose the smallest key which provides a reasonable level of security. The actual level of difficulty in increasing key length varies with the encryption method and its implementation. In hardware, obviously, it is impossible to avoid the fact that a larger key will require more physical wires in at least some parts of the system. In software, some algorithms (such as RSA, where larger primes mean more computation) increase in complexity significantly with key size, while others (including many symmetric block ciphers) increase in complexity only slightly with larger keys. At worst, since the difficulties associated with larger keys tend to grow linearly, or at least far less than exponentially, with key size, it is relatively easy to achieve a key size which puts cracking well outside the range of feasibility. As computing power, as well as cryptographic and cryptanalytical techniques grow, of course, it will be necessary for crypto systems and key sizes to grow with them in order maintain their level of security.

The other limitation on key length is societal, in the form of the limitations placed on encryption by governments and law-enforcement agencies who wish to ensure their continued ability to bypass that encryption. Those limitations, and the various proposed methods for eliminating them, have been and will continue to be the subject of voluminous debate by people from all sides and with all different priorities and points of view. If encryption is to be a tool for security and trust on the computer networks of the future, then those debates must eventually reach conclusions which allow encryption to be secure, and do not present new vulnerabilities or technical complications which cripple the encryption systems in question.

Conclusions

The potential risks demonstrated by distributed responses to encryption challenge go beyond their intended goal of pointing out inherent shortcomings in encryption systems. They also demonstrate a real and viable method by which those encryption systems can be attacked, to devastating effect. Though the public distributed efforts have been dedicated to useful purposes, there is a strong possibility of similar methods falling into the hands of individuals or organizations with other intentions, and it is quite possible for those individuals to appropriate the use of distributed computing power without the knowledge or consent of the owners or operators of the computer systems in question. Though the sheer number of cracks possible by distributed efforts over a period of time is relatively small, the encryption and security systems of today contain weak points and single keys of enough importance that those limited resources, if well directed, could lead to serious breaks into those encryption systems, with the large potential losses of data, money, or even property that can result. The threats demonstrated by distributed cracks must be addressed and minimized or eliminated if computers, networks, and society's dependence on them are to continue to grow. Though solutions exist, they are not all simple due to the tradeoffs involved, and the fact that distributed computing itself has great useful potential which cannot be allowed to be eliminated by fears of its misuse. The architects of the computer networks of tomorrow will need to ensure that such threats are kept in mind, and dealt with so that those networks can be trusted as venues for the daily business of society.

Sources

Unpublished web references are not given a date, but were referenced in their most up-to-date form as of January 1998.

[1] RSA Laboratories: The RSA Data Security Secret-Key Challenge,

[2] RSA Laboratories: DES Challenge II, .

[3] Rocke Verser: The DESCHALL II Homepage, , and original DESChall Homepage, .

[4] Distributed.Net Homepage, , which includes sub-pages for the DES Challenge II (at /des/) and RC5 Challenge (at /rc5/).

[5] Mersenne.Org (mathematical problems),

[6] Ben Adida & Oliver Roup, RC5 Java Breaking Effort, .

[7] Email exchanges with Ben Adida (ben@mit.edu), January 1998.

[8] Multiple Authors, _The Risks of Key Recovery, Key Escrow, and Trusted Third-Party Encryption_, May 1997, <http://www-swiss.ai.mit.edu/6805/articles/crypto/key-study-report.html>.