CS427 Project

SOCat Cryptography Analisis

CS427 Project

Winter 2016

Student:
Sam Quinn
Quinnsa@Oregonstat.edu

Professor:
Michael Rosulek
rosulekm@eecs.oregonstate.edu

March 15, 2016

Contents

1 Introduction

One common problem with networked computers is how to transfer files between them in a secure way. There are a few solutions out there, many secure and there are many that are not. In this analysis I am going to talk about how one particular mistake in Socat, a common Unix based network transfer application, compromised their entire security.

Socat is an application that can concatenate program output or files to a network socket. Socat can relay data in a bidirectional manner between two independent data channels. Socat can transmit to a file, pipe, device, or as socket (IP4, IP6, UDP, TCP) all encrypted over SSL. It is easy to see the need for SSL here since anytime you send data over a network any eavesdropper can view your data in plaintext, unless however it is encrypted.

2 Problem Defined

Socat secures network transmission with the extremely popular OpenSSL. OpenSSL is in fact still considered very secure, Socats problems were products of their own misuse of the secure libraries within OpenSSL. Socat has implemented a hybrid encryption scheme to securely transfer data among peers. The symmetric keys are obtained using Diffie Hellman key agreement.

Diffie Hellman protocol needs to publicly share two variables g and p to function correctly. The variable p is public and is very large since the p value will be used as the modulus for the combined shared key after the two parties raise the g to their private keys. When generating p and g you want to choose good numbers. For example p that is greater than $2048bits$ in length and that satisfy $p = 2t + 1$ where t is also prime are know to be harder to break. When choosing p values the most important thing is that both p and g **MUST** be prime, which is where Socat went wrong. Socat for a full year had a p value that was not prime.

The Diffie Hellman p that they used for almost a year was:

```
static unsigned char dh1024_p[] = {
   143319364394905942617148968085785991039146683740268996579566
   827015580969124702493833109074343879894586653465192222251909
   074832038151585448034731101690454685781999248641772509287801
   359980318348021809541131200479989220793925941518568143721972
   993251823166164933334796625008174851430377966394594186901123
   322297453
 }
```

I do give it to the developer that the original number is not exactly easy to tell if it is prime or not, But given how easy it is to verify prime numbers, it should have been checked. OpenSSL has even made the process of generating Diffie Hellman primes easier by including the dhparam function which given the -C flag will actually output c code that the developer could have seriously just copy and pasted into Socat.

While I was reading into how this non-prime number got here in the first place it began to

smell very fishy, how could the developer at Socat mess this up unless he was trying too? The development of Socat is open source so I went back to the exact commit message when this non-prime value was added.

```
--- a/CHANGES
+++ b/CHANGES
@@ -72,6 +72,9 @@ corrections:

       fixed a few minor bugs with OpenSSL in configure and with messages

+      Socat did not work in FIPS mode because 1024 instead of 512 bit DH prime
+      is required. Thanks to Zhigang Wang for reporting and sending a patch.
+
```

The commit where the non-prime was added was to replace the first 512*bit* prime number due to an incompatibility with FIPS (Federal Information Processing Standard). On January 4 2015 Zhigang Wang, an Oracle Solution Specialist reported the FIPS incompatibility and a submitted the patch changing the 512*bit* prime to the 1024*bit* number. This commit message was made by Gerhard Rieger the maintainer and lead developer but clearly sited Zhigang Wang for the patch.

3 Why the problem is bad

The first p that was actually prime was pretty small, and did not adhere to the FIPS standard and was much below the recommend minimum Diffie Hellman prime size.

3 According to NIST SP 800131A, DiffieHellman with 1024bit key sizes are deprecated through the end of 2013 and 2048bit key sizes will become the minimum. Users should start transitioning to the larger key sizes. [1].

This gives some credit back to Zhigang Wang for responsibly telling the Socat's developer this information but if Zhigang Wang was the one to actually submit the non-prime p, what could he have actually done with it.

Lets first look at what the p has to do with everything. Size does matter with the public prime numbers, small primes make it easier for an adversaries to attack the key agreement. The first p was 512*bits* which is not considered secure anymore, mainly because there are machines out there that could attack this given their computational power. If Zhigang Wang wanted to hack Socat it probably would have been easier not submitting the patch for the larger prime p. Lets say that this number was submitted for the purpose of creating a backdoor.

You might wonder why having a non-prime p could do to hinder the security of the Diffie Hellman key agreement, p is public, everyone already knows it. The adversary is trying to solve the problem given g^a and g^b what is the value of $g^a b$ is. This equation is also known as the Diffie Hellman Problem or more generic discrete logarithm problem. It is true that p is public, but still calculating discrete logs takes a lot of time and computational power. One math trick that can speed up modulus calculations is the Chinese Remainder Theorem (CTR). The CTR states

that given any mathematical computation within $\mathbb{Z}_p$ the same calculation can be calculated $modp$'s factors. With our modern computers we are able to perform calculations much easier on the smaller mod factors of p.

The Chinese remainder theorem is a very powerful function in cryptanalysis. When number can be broken into smaller prime factored parts the number is considered smooth. The discrete logarithm problem can be solved in the smaller groups and then use the CTR to obtain the discrete log of g^{ab}. The current best algorithm that will solve the discrete logarithm problem is Number Field Sieve which holds the current record for a random modulo p that is 530 bits long. So does a non-prime number help in bruteforcing the DLP? It turns out not really.

3.1 Backdoor Possibilitties

How an adversary could attack the DHKA would be with Pohlig hellman attack.

To fully execute a Pohlig-Hellman attack we need to find $\Phi(p)$ also known as the Euler totient of the Modulus. Even if we separate the discrete logarithm problem into smaller subsets using the CTR, the size of the factors will still affect the difficulty of the DLP of the multiplicative group $\mathbb{Z}_p$. So far given the insecure prime p used in Socat the factors that have been found so far are $a = 271$, $b = 13587$, and
$c = 38894884397634366007356454548332370646972724268802781973440208895542936165564673524541403310393405820598366261673173802130771236325314878371830363723788045821711985461441675679316058246609104355161134470046705337593170498462616195650378975298117141144096886684800236261920005248055422089305813639519$
c is a $1002bit$ non-prime number that we are able to further factorize, but as for now the factors remain unknown. With trial division we are able to determine the the smallest factors are larger than 2^40. Trial division is a test where it will systematically test whether p is divisible by any smaller number, clearly very slow as it checks every number in a Sieve of Eratosthenes like manner. If we want to get anywhere near the factors of c we will have to do something more efficient like the Lenstra Elliptic Curve factorization. Lenstra algroithim's computation performance is a based on the size of the largest factor not the entire number itself. However since there are still no solutions the factors of c might be a product of two large $500bit$ primes, which would be still very difficult to solve and may never be found. With still no solutions of the factors of c we are still not able to take the full advantage of the CTR in computing the DLP.
The computations that would need to take place after the smaller factors of c are found are with the Chinese remainder theorem. If the subsets were small enough we would be able to brute force x_1 and x_2 but if they are still too large we could use more efficient DLP algorithms. DLP can be solved more efficiently with either Pollar's Rho or index calculus in each of the smaller sub groups. For example:

$\Phi(p) = q_1 * q_2$

find x_1 such that $h^{q_1} = (g^{q_1})^{x_1}(modp)$

find x_2 such that $h^{q_2} = (g^{q_2})^{x_2}(modp)$

Then we can prove that $x = x_1(modq_2)$ and $x = x_2(modq_1)$, and thus can figure out x with the CRT.
If an attacker could find x_1 and x_2 then they would be able generate the same key as the secured participants would have generated and be able to decrypt any message they passed between them, in this case it would have been the symmetrical key. This would give the adversary full access to any data sniffed over the wire [2].

As for now there have not been any known successful attacks on Socat due to the p not being prime. It is also clear to see that even though the p was not prime it still was very hard for any adversary to solve the DLP to launch an attack. However, it is still unclear how that number was put into place an could have been precalculated with certain factors that only the Zhigang Wang knew.

4 How they fixed the problem

This problem was discovered by Santiago Zanella-Beguelin who is apart of Microsoft Vulnerability Research (MSVR) group. Microsoft will never reveal vulnerability details before a vendor-supplied update is available for issues reported though the MSVR program unless there is significant evidence of active attacks in the wild. [MSVR]. Microsoft Vulnerability Research discloses vulnerabilities to the third-parties with a Coordinated Vulnerability Disclosure (CVD) approach [3].
The fix for Socat was simple generate a new Diffie Hellman p parameter, and update it to the new FIPS standard of 2048-bits. While it is unclear if the non-prime value was actually ever exploited or put in place for a malicious backdoor. With the uncertainty if the non-prime value was a mistake or not makes the analysis of security guarantees are hard to determine, if the value was set in place as a backdoor then it very well could have been exploited.
Zhigang Wang was asked to comment on how the non-prime number that was submitted as a patch was created, to which no response was sent back.

5 Conclusion / What I learned

To exploit the bug that was in the Socat DHKA you must do the following:

1. Factor p to smaller values, most efficient would be Lenstra elliptic curve.
2. Use Pohlig-Hellman and CRT to reduce DLP to the factor sets of $\mathbb{Z}_p$
3. Use Pollar's rho or index calculus to solve DLP in each factor set of $\mathbb{Z}_p$
4. Sniff Socat encrypted network traffic to recover public DH parameters
5. Unencrypt data

I also took note on the fact that you should always double check your security implementation, or even have another person look it over to find stupid errors like a prime not being prime. It is also an I opener to how long it might be before someone finds the mistake in your code, and that hole time you should assume that an adversary has known about it the whole time and consider all data that was used with that application compromised. It was fun for me to look into this topic, I really liked how it was a mystery why the value was not prime and I came up with multiple conspiracy theories while writing this to why the non-prime value was added. It still does not seem like a mistake but at this point I might just have to leave it at that, a mistake.

References

[1] "Security policy for fips 1402 validation." Enhanced DSS and DiffieHellman Cryptographic Provider.

[2] suy ngh th nghim, "Exploiting the diffie-hellman bug in socat," 2016.

[3] "Microsoft vulnerability research (msvr)." Security TechCenter.