null – The Open Security Community

Learning from the Past

Cryptography Vulnerabilities Exploited in 2020

White Paper

Table of Contents

1 About null 3

2 Acknowledgment 3

3 Abstract 4

4 Motivation 6

5 Overview 6

6 Vulnerability Details 7

6.1 Side Channel related attacks 7

6.1.1 Raccoon Attack reveals Premaster Secret of DHE in older TLS 7

6.1.2 CVE-2020-25659 Python RSA timing flaw 9

6.1.3 CVE-2020-8694/95: Intel SGX Side Channel attacks 14

6.1.4 LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage 17

6.2 Certificates related flaws 20

6.2.1 GitHub homepage down due to expired certificate 20

6.2.2 CVE-2020-0601: Flaw in Windows CryptoAPI (Crypt32.dll) implementation of Elliptic Curve Cryptography (ECC) can result in certificate forgery 23

6.2.3 Certificate Expiration Takes Down Spotify 25

6.3 Buffer overflow 27

6.3.1 “pool-based” buffer overflow vulnerability in the Windows Kernel Cryptography Driver 27

6.3.2 CVE-2020-36242 - Python crypto lib 30

6.4 Certifying Authority Best Practices gap 32

6.4.1 Let's Encrypt has to revoke 3 Million certificates due to CAA Rechecking bug 32

6.4.2 Violation of the Baseline Requirements by Microsoft’s intermediate CA platform while generating OCSP responder certificate 34

6.5 Other Categories 36

6.5.1 'Zerologon' Vulnerability in Netlogon Could Allow Attackers to Hijack Windows Domain Controller 36

6.5.2 Identrust OCSP verification was failing 41

6.5.3 TOR nodes stripped TLS off of traffic going to some Bitcoin sites 44

6.5.4 wolfSSL TLS 1.3 Client Man-in-the-Middle Attack CVE-2020-24613 47

6.5.5 CVE-2020-1971: NULL pointer dereference bug in OpenSSL cause crash while verification of certificate. 51

6.5.6 CVE-2020-0688: Remote Code Execution on Microsoft Exchange Server through fixed cryptographic keys 54

6.5.7 CVE-2019-14855: Chosen-prefix collision for SHA-1 56

6.5.8 CVE-2020-2655 JSSE Client Authentication Bypass 59

6.5.9 Netgear Signed TLS Cert Private Key Disclosure 62

6.5.10 CVE-2020-8172: Node.js - Insecure TLS session reuse can lead to hostname verification bypass 66

6.5.11 CVE-2020-13777: GNU TLS 1.3 session resumption works without a master key, allowing MITM 69

6.5.12 Go Crypto libraries panic during the recursive division of very large numbers CVE-2020-28362 71

6.5.13 SSRF attack using TLS features, research presented at DEFCON safe mode 73

6.5.14 CVE-2020-11651,52 Allows Unauthorized Access to SaltStack Master Server 76

1 About null

null - The open security community is one of the most active and vibrant communities of cybersecurity professionals. Started with the simple idea of providing a knowledge-sharing platform to cybersecurity professionals, null has grown many folds. Currently, null has 20+ active chapters and organizes many security events for aspiring cybersecurity professionals. null is about spreading information security awareness. All our activities such as null Monthly Meets, null Humla, null Bachaav, null Puliya, null Job Portal are for the cause of that.

null is now focusing on contributing to enterprise security. Working on many projects to collaborate with the enterprise, such as:

- Defining security frameworks and standards for producing security guidelines in the upcoming IT technology like Cloud, Blockchain/Cryptography, IoT, AI/ML, and many more.
- Develop tools and methodologies in order to secure the products and infrastructure based on the above-mentioned technologies.
- Start many new security projects and publish research papers.

This white paper is one small effort to our contribution to achieving the above objectives.

2 Acknowledgment

On behalf of **null - The open security community**, we would like to thank the authors of this white paper, who have contributed their precious time and effort to publish this paper.

Authors

- Ajit Hatti (https://www.linkedin.com/in/ajithatti/)
- Praneeth Varma (https://www.linkedin.com/in/v-s-k-p-v-0247b120/)
- Sagar Yadwad (https://www.linkedin.com/in/sagar-yadwad-79711123)
- Sumanth Thyagarajan (https://www.linkedin.com/in/sumanth-thyagarajan/)
- Bhavesh Dhake (https://www.linkedin.com/in/bdhake/)

Project Coordinator

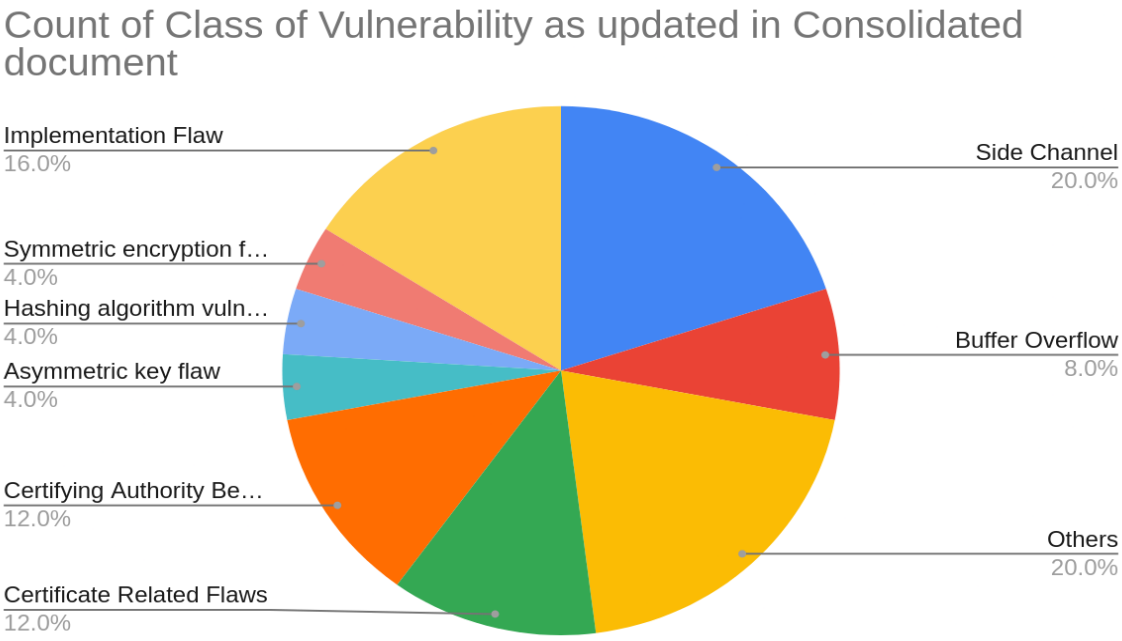
- Murtuja Bharmal (https://www.linkedin.com/in/murtujabharmal/)

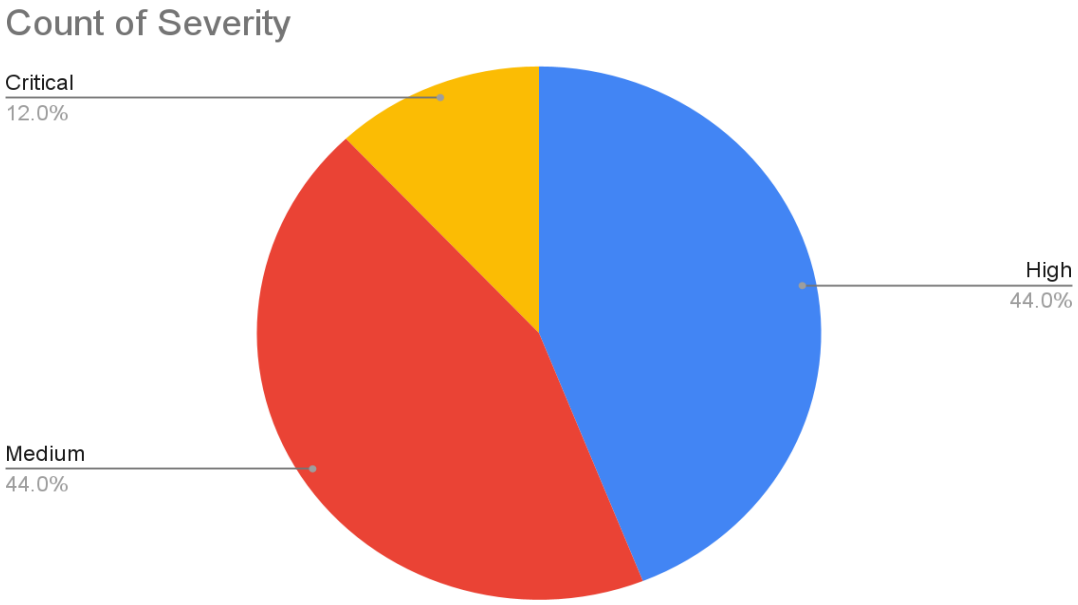
3 Abstract

The year 2021 was rocked by many incidents which proved the academic theories and decade-old mathematical predictions right. The Chosen-prefix collision for SHA-1 was published by researchers Gaëtan Leurent and Thomas Peyrin in January 2020.

The majority of incidents we saw were related to improper use and maintenance of domain certificates. Let's Encrypt came close to revoking 3 Million certificates due to the CAA Rechecking bug, which was the hallmark of certificate-based incidents this year.

We also saw many implementation flaws appearing from time to time throughout 2021. Zerologon' Vulnerability in the Netlogon service, which can allow Attackers to Hijack Windows Domain Controllers, was widely exploited even after the release of the patch from the Vendor.





TLS 1.3 has gained wide adoption in the last few years, with a promise to make the internet more secure and free from common network-based attacks. We saw 3 vulnerabilities arising due to implementation flaws of TRLS 1.3. CVE-2020-13777 and SSRF attack show more refinement is needed to TLS 1.3 specification as it becomes mainstream.

Compared to last year, we have seen lesser incidents involving Certifying Authority (80% reduction). Most incidents we saw were implementation flaws, which proves the need for better education awareness among the developers and implementers of cryptographic libraries.

As the community struggles to control the malicious nodes in the ToR project (TOR nodes stripped TLS off of traffic going to some Bitcoin sites.) we look forward to better answers to keep the project ToR more resilient and in control to firmly handle rogue elements causing serious harm to ToR users. We hope to see a cryptographic solution for this human problem soon.

Null Cryptographic Club has taken year-long efforts to make this report. We welcome your feedback and suggestions.

Till then, from all of us here at null family, stay safe & Love Cryptography.

4 Motivation

Cryptography being a foundational discipline in cybersecurity, the discussion around its vulnerabilities, good practices standardization in practices is left to the cryptographer who is not the practitioners. Practitioners of cybersecurity are focused on all conventional vulnerabilities except for cryptographic vulnerabilities.

This paper is an attempt to address the gap and initiate discussion & interest of the security community in the wider realm to adopt and practice the affinity for cryptography & its secure implementation.

Leveraging the forum of the null open security community, we attempt to analyze the cryptographic vulnerabilities of 2020, which have been impacting the year 2021, and we expect them to continue to do the same until we seriously pay attention to better practices and procedures when it comes to cryptography implementation and securing the cryptographic resources.

5 Overview

We have collected and organized most of the cryptography implementation-related vulnerabilities which are exploited in 2020. We have classified these vulnerabilities into below mentioned major categories:

1. Flaws in cryptographic schemes
2. Information disclosure
3. Implementational flaws
4. Side Channel
5. Certificate Related Flaws
6. Buffer Overflow
7. Certifying Authority Best Practices gap
8. Others.

For all the vulnerabilities, we have provided the following details:

1. Vulnerability description
2. Severity
3. How it can be exploited
4. Impact
5. How to mitigate it

A brief about each vulnerability can be found in the below section.

6 Vulnerability Details

6.1 Side Channel related attacks

6.1.1 Raccoon Attack reveals Premaster Secret of DHE in older TLS

Introduction

Raccoon is a timing attack affecting older TLS versions that uses Diffie-Hellman key exchange (DHE).

Servers commonly reused the DH key exchange parameters to negotiate pre-master secrets. TLS 1.2 and versions before, prescribed to strip the leading zeros of premaster secret before it is used to generate a master secret from a key derivation function. This stripping of leading '0' introduces a time difference between pre-master-secrets with and without leading zeros.

An attacker can use this as an oracle to deduce the master secret and decrypt the TLS communication between client and server.

The older versions of F5's BIG-IP platforms with Cavium Nitrox SSL hardware acceleration cards, a virtual server configured with a Client SSL profile were found vulnerable to Raccoon attack for which F5 released a CVE-2020-5929

CVSS Score: 7.4 / High

Description

The research paper Raccoon Attack: Finding and Exploiting Most-Significant-Bit-Oracles in TLS-DH(E) by Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk give the complete technical details of the vulnerability.

We share the overview to give readers a quick understanding of the issue with the help of extracts from their writeup.

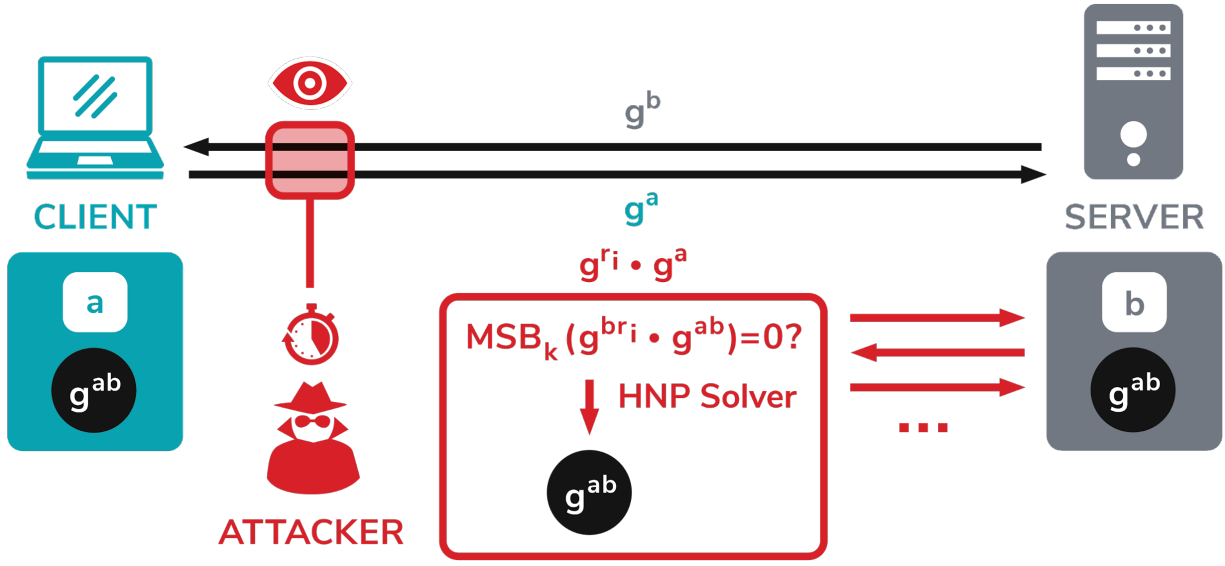
TLS 1.2 (and all previous versions) prescribes that all leading zero bytes in the premaster secret are stripped before used in further computations. Since the resulting premaster secret is used as an input into the key derivation function, which is based on hash functions with different timing profiles, precise timing measurements may enable an attacker to construct an oracle from a TLS server.

Exploitation

As described above, the varying length of pre-master secret depending on the leading '0's introduces the timing attack as the varying length of data is processed in different numbers of blocks by hash functions in different amounts of time.

This becomes a timing oracle to detect leading '0' or 0s in a pre-master secret in a TLS handshake. Learning one byte from a premaster secret would not help the attacker much. However, here the attack gets interesting. Imagine the attacker intercepted a ClientKeyExchange message containing the value g^a.

The attacker can now construct values related to g^a and send them to the server in distinct TLS handshakes. More concretely, the attacker constructs values $g^{r_i}*g^a$, which lead to pre-master secrets $g^{r_i*b}*g^{ab}$.



Based on the server timing behavior, the attacker can find values leading to pre-master secrets starting with zero. In the end, this helps the attacker to construct a set of equations and use a solver for the Hidden Number Problem (HNP) to compute the original premaster secret established between the client and the server.

The attack is also exploitable without the complex timing analysis if the server reuses DH public keys. The authors found a good number of servers on the internet reuse the DHE keys.

Impact

Successful exploitation of Raccoon attack allows an attacker to decrypt the connection between users and the server.

All the confidential data exchanged over such encrypted channels like passwords, credit card numbers, emails, instant messages, and sensitive documents are exposed.

The vulnerability was rated as a High Impact vulnerability with the CVSS score of 7.4

Mitigation

At Null cryptography group we recommend administrators to configure their servers to use the latest Transport Layer Security protocol version which is TLS1.3 at this point of time.

The other precautions administrators can take is to configure servers not to reuse Diffie-Hellman key exchange parameters. The elliptical curve Diffie-Hellman key exchange (ECDHE) is not affected by Raccoon but is not Quantum Cryptanalysis safe.

For future purposes we have to wait and see for the recommended key exchange mechanism which can be a combination of the ones we have been using or new quantum safe exchange schemes are proposed.

References

1. https://raccoon-attack.com/
2. https://raccoon-attack.com/RacoonAttack.pdf
3. https://support.f5.com/csp/article/K91158923

6.1.2 CVE-2020-25659 Python RSA timing flaw

Introduction

CVE-2020-25659 Python cryptography library versions between 1.5 to 3.2 were found to have vulnerable RSA decryption API, which leads to information disclosure

A flaw was found in python-cryptography, between version 1.5 to 3.2, where RSA decryption API is vulnerable to Bleichenbacher timing attacks. This flaw allows an attacker, to decrypt parts of the ciphertext encrypted with RSA, PKCS#1 v1.5 format.

The highest threat from this vulnerability is confidentiality.

CVSS Score: 5.9 / Medium

Description

A security flaw was found in Python in rsa.py file, where RSA encrypted content when verified for PKCS#1v1.5 format gives descriptive errors which can be exploited to launch timing/side channel attacks to disclose confidential information.

The change logs on github shows the descriptive error provided in the code before the fix.

```
124 -         _handle_rsa_enc_dec_error(backend, key)
125 -
126 -     return backend._ffi.buffer(buf)[: outlen[0]]
127 -
128 -
129 - def _handle_rsa_enc_dec_error(backend, key):
130 -     errors = backend._consume_errors_with_text()
131 -     if isinstance(key, _RSAPublicKey):
132 -         raise ValueError(
133 -             "Data too long for key size. Encrypt less data or use a "
134 -             "larger key size.",
135 -             errors,
136 -         )
137 -     else:
138 -         raise ValueError("Decryption failed.", errors)
```

Change log for fix of CVE-2020-25659

In the partial fix provided by the developers, the error message was changed from a descriptive one to a generic one.

```
131 -     if isinstance(key, _RSAPublicKey):
132 -         raise ValueError(
133 -             "Data too long for key size. Encrypt less data or use a "
134 -             "larger key size.",
135 -             errors,
136 -         )
137 -     else:
138 -         raise ValueError("Decryption failed.", errors)
```

Before the fix

```
133 +         raise ValueError("Encryption/decryption failed.")
134 +     return resbuf
```

After the fix

Exploitation

Exploiting Timing Attack Side channel vulnerabilities

In any programming language many branching, looping, decision statements are taken after comparison of values of 2 string values.
Commonly such comparison happens character by character and as soon as the first different character is found between the 2 strings, the control switch happens to the next statement, which is generally an error message or an outcome different than what is expected on complete matching of the 2 strings.

Sooner the difference is found, quicker is the control switch, and later in the comparison the difference between the 2 strings is found, later the control switch happens.

In a scenario where an attacker has to guess a secret value, say password to login into a system, if the string provided by an attacker is compared byte-by-byte with the stored value of password, the attacker can leverage timing attack.

Attack Steps

Assuming the password length is 10 characters, the attacker sends a 20 characters string initialized with all zeros. This obviously not being a valid password, the system will send an error. Attacker notes the delay in receiving the error message.

Then the attacker increments the first character keeping all the remaining values, the same and sends it to the server and notes the delay in response. If the response time is the same, the attacker goes on to increment the value of the first character till he sees an increase in delay.

Password Guess	delay	observation
00000000000000000000000000000000000000000000	- 200 ms	no match
01000000000000000000000000000000000000000000	- 214 ms	no match
02000000000000000000000000000000000000000000	-195 ms	no match
... snip 250 ...		
A1000000000000000000000000000000000000000000	- 205 ms	no match

A2000000000000000000000000000000000000000	- 200 ms	no match
A3000000000000000000000000000000000000000 match found	- 380 ms	<<< 1st character
A4000000000000000000000000000000000000000	- 208 ms	no match

At this point the attacker got the match of the first character. Now the attacker moves on to increment the second character till he finds further delay in response time.

In this way the attacker goes on to guess all the remaining characters.

Efficiency

This method is far more efficient than brute-forcing the entire character space which is not practical. Learn more about timing attacks here.

Impact

Timing attacks can be used to disclose secret strings, session keys, encryption keys, passwords, pins, etc. For the CVE-2020-2569 the Impact was assessed by Red Hat Inc as **Medium** with a CVSS score of **5.9**

Mitigation

At Null cryptography group we recommend the following best practices for developers to prevent timing attacks.

1. Use generic Error message

Any function returning an error message which can give knowledge about the exact reasons of failure, format errors, padding errors, length difference or any inconsistency acts as an oracle and can be used by an attacker to deduce the confidential information.

To avoid oracle-based attacks avoid providing error messages with extra details.

2. Time safe coding

To write a code which is secure from timing attacks, we recommend using “Constant Time Comparison Functions”. OpensSSL provides the CRYPTO_memcmp() function for this.

```
int CRYPTO_memcmp(const void *in_a, const void *in_b, size_t len)
    {
    size_t i;
    const unsigned char *a = in_a;
    const unsigned char *b = in_b;
    unsigned char x = 0;

    for (i = 0; i < len; i++)
        x |= a[i] ^ b[i];

    return x;
    }
```

3. Be aware of the compiler optimizations

Many time compiler optimizations convert constant time comparison functions to non-constant time comparison functions. Consider OpenBSD's timingsafe_bcmp():

```
int
timingsafe_bcmp(const void *b1, const void *b2, size_t n)
{
    const unsigned char *p1 = b1, *p2 = b2;
    int ret = 0;

    for (; n > 0; n--)
        ret |= *p1++ ^ *p2++;
    return (ret != 0);
}
```

The above function is a constant time comparison function and it can get optimized by compiler in a following way making it non-constant time comparison function

```
int
timingsafe_bcmp(const void *b1, const void *b2, size_t n)
{
    const unsigned char *p1 = b1, *p2 = b2;
    for (; n > 0; n--)
        if (*p1++ ^ *p2++)
            return 1;  // Not constant time.
    return 0;
}
```

4. Compare Hashes of the values instead of the actual values

When uncertain of writing an appropriate constant-time comparison function and what the compiler optimization can do to it we recommend comparing hashes instead of actual values.

For example, for verification of a message authentication code (mac) of a message (msg), encrypted with key (key), and signature (S) sent by the remote user the usual way of verification is like this

```
def verify ( key , msg, sig)
        mac = HMAC( key, msg)
        return ( mac == sig )
```

Instead of this do this

```
def verify ( key , msg, sig)
        mac = HMAC( key, msg)
        return (  HMAC(key, mac)  == HMAC(key, sig) )
```

Since hash of partially matching or even slightly strings are also radically different, timing attacks won't work in this case.

References

1. https://github.com/pyca/cryptography/pull/5507/commits/ce1bef6f1ee06ac497ca0c837fbd1c7ef6c2472b
2. https://access.redhat.com/security/cve/cve-2020-25659
3. https://github.com/nodejs/node-v0.x-archive/issues/8560

6.1.3 CVE-2020-8694/95: Intel SGX Side Channel attacks

Introduction

A side-channel attack is any attack based on information gained from the behavior of a computer system, rather than finding a weakness in the algorithm. The information can be - time taken, power consumption, electromagnetic leaks, sound, and all other sources that can provide an extra source of information, which can be exploited.

It was found that the Intel SGX enclaves were compromised and many side-channel attacks were found on the architecture.

CVSS Score: 5.5 / Medium

Description

Intel SGX is a set of security-related instruction codes that allows user-level code to allocate private regions of memory, called enclaves, which are designed to be protected from processes running at higher privilege levels.

PLATYPUS, A software-based power side-channel attack on Intel server, desktop and laptop CPUs. This can be achieved by exploiting unprivileged access to the Intel RAPL interface exposing the processor's power consumption to infer data and extract cryptographic keys.

CacheOut, a speculative execution attack that is capable of leaking data from Intel CPUs across many security boundaries. An attacker can exploit the CPU's caching mechanisms to select what data to leak, as opposed to waiting for the data to be available. The data can be leaked from the OS kernel, co-resident virtual machines, and even SGX enclaves.

SGAxe, is an evolution of CacheOut, specifically targeting SGX enclaves. Despite extensive efforts done by Intel to mitigate SGX side channels, an attacker can still breach the confidentiality of SGX enclaves even when all side-channel countermeasures are enabled. The attacker can perform extraction of SGX private attestation keys from within SGX's quoting enclave, as compiled and signed by Intel. With these keys in hand, the attacker signs fake attestation quotes, just as if these have been initiated from trusted and genuine SGX enclaves. This erodes trust in the SGX ecosystem, as using such quotes, an attacker can masquerade itself as a genuine SGX enclave to a remote party while offering little protection, in reality.

Exploitation

PLATYPUS

- An adversary typically attaches an oscilloscope to monitor the energy consumption of a device. The Intel Running Average Power Limit (RAPL) interface allows monitoring and controlling the power consumption of the CPU and DRAM in software. The CPU comes with a power meter. With the current implementation of the Linux driver, every unprivileged user has access to its measurements.
- PLATYPUS tool can detect electrical signals and find secrets in the processor's energy measurements using Intel RAPL.
- The variations in the power consumption are observed to distinguish different instructions and different Hamming weights of operands and memory loads, allowing inference of loaded values. The values are filtered using a running average, making it harder to infer secrets.

- PLATYPUS can further infer intra-cache line control flow of applications, break KASLR, leak AES-NI keys from Intel SGX enclaves and the Linux kernel, and establish a timing-independent covert channel.
- PLATYPUS with precise execution control of SGX-Step. The RSA keys are processed by mbed TLS from an SGX enclave.

SGAxe

- Victims' data is filled in cache and fill the buffer, however verw flushes that data from the fill buffer
- The cache is divided into sets, the address determines which set the data is and the attacker reads the mapping to the same set and fills the cache set with its data.
- Evicting the victim’s data into the fill buffer while the data gets written back to DRAM
- The attacker uses the faulty load to leak the data
- First by extracting the SGX private attestation keys from within Intel SGX’s quoting enclave
- These keys are compiled and signed by Intel itself
- The next step includes signing arbitrary SGX attestation quotes and faking them to look like they been initiated from trusted and updated SGX enclaves
- This allows the network attacker to masquerade as authentic SGX Intel machines

Impact

SGAxe

- First/Second generation Intel® Xeon® Processor Scalable Family-based on Skylake/Cascade Lake microarchitecture
- 6th generation Intel® Core™ processors and Intel® Xeon® processor E3-1500m v5 product family and E3- 1200 v5 product family based on Skylake microarchitecture
- 7th/8th generation Intel® Core™ processors based on Kaby/Coffee Lake microarchitecture
- 7th/8th generation Intel® Core™ processors based on Kaby/Coffee Lake microarchitecture
- Coffee Lake
- 8th generation Intel® Core™ processors based on Whiskey Lake(ULT)
- Whiskey Lake (ULT refresh)
- Whiskey Lake (Desktop)
- 10th Generation Intel® Core™ processors based on Amber Lake Y

PLATYPUS

- 8th Generation Intel® Core™ Processor Family
- 10th Generation Intel® Core™ Processor Family
- 9th Generation Intel® Core™ Processor Family
- Intel® Xeon® Processor E Family
- Intel® Pentium® Processor Silver Series Intel® Celeron® Processor J Series Intel® Celeron® Processor N Series
- Intel® Xeon® Processor E3 v6 Family

Mitigation

- The Intel SGX vulnerabilities are tracked as CVE-2020-8694 & CVE-2020-895. Patches for these vulnerabilities are released, applying the patch can fix this vulnerability.

References

- https://www.intel.in/content/www/in/en/architecture-and-technology/software-guard-extensions.html
- https://news.hackreports.com/sgaxe-crosstalk-attacks-intel-sgx-vulnerability/
- https://www.csoonline.com/article/3596564/intel-sgx-users-need-cpu-microcode-patch-to-block-platypus-secrets-leaking-attack.html
- https://arstechnica.com/information-technology/2020/03/hackers-can-steal-secret-data-stored-in-intels-sgx-secure-enclave

6.1.4 LadderLeak: Breaking ECDSA With Less Than One Bit Of Nonce Leakage

Introduction

The ECDSA algorithm is one of the most widely deployed signature schemes today and is part of many practical cryptographic protocols such as TLS and SSH. Its signing operation relies on an ephemeral random value called a *nonce*, which is particularly sensitive: it is crucial to ensure that the nonces are kept in secret and sampled from the uniform distribution over a certain integer interval. It is easy to see that if the nonce is exposed or reused completely, then an attacker is able to extract the secret signing key by observing only a few signatures. By extending this simple observation, cryptanalysts have discovered stronger attacks that make it possible to recover the secret key even if short bit substrings of the nonces are leaked or biased.

CVSS Score: 5 / Medium

Description

To deliver the high performance expected of modern computers, processors employ an array of techniques that aim to predict program behavior and optimize the processor for such behavior. Asa direct consequence, program execution affects the internal state of the processor, which in turn affects the speed of future program execution. Thus, by monitoring its own performance, a program can learn about the execution patterns of other programs, creating an unintended and unmonitored communication channel.

Unintended leakage of cryptographic software execution patterns can have a devastating impact on the security of the implementation. Over the years, multiple attacks have demonstrated complete key recovery, exploiting leakage through the internal state of various microarchitectural components, including caches, branch predictors, functional units, and translation tables.

Flush+Reload is a prominent attack strategy, in which the attacker monitors victim access to a memory location. The attack consists of two steps. In the flush step, the attacker evicts the monitored memory location from memory, typically using a dedicated instruction such as CLFLUSH. The attacker then waits a bit to allow the victim time to execute. Finally, in the reload step, the attacker accesses the memory location, while measuring how long the access takes. If the victim has not accessed the monitored location, it will remain uncached, and access will be slow. Conversely, if the victim has accessed the monitored location between the flush and the reload steps, the memory location will be cached, and access will be fast. Repeating the attack, an attacker can recover the memory usage patterns of the victim, allowing attacks on symmetric ciphers, public key primitives, both traditional and post-quantum, as well as on non-cryptographic software.

Exploitation

The cache-timing attacks were implemented using Flush+Reload from the FR-traceprogram available in the Mastik side-channel analysis toolkit. OpenSSL was targeted by running the command-line signature computation in two Broadwell CPUs. OpenSSL was built using a standard configuration with debugging symbols and optimizations enabled. Targeted parameters were at the lowest security in each class:sect163r1for the binary and P-192/secp192k1 for the prime case. The timing difference for computing the first iteration of the ladder was amplified to around 100,000 and 15,000 cycles for the binary and prime case, respectively. Amplifying the timing difference made it feasible to detect the second MSB with high probability: around 99% for curvessect163r1and P-192. the slot was configured, or the time between two consecutive probings by the Flush+Reload thread, to 5,000 cycles to obtain finer granularity. In the binary case, the detection strategy consisted of first locating in the traces a cache-hit matching the execution of the first field multiplication by Z1ingf2m_Madd()at the beginning of the first ladder iteration, and then looking for the

next cache-hit matching the second field multiplication which marks the end of the first. If the number of slots between the two was above 15, this means a timing difference of at least 75,000 cycles. The first version of the attack achieved 97.3% precision, which was later improved. When running the attack against 10,000 signature computations, we were able to correctly detect 2,735 signatures with second MSB 1 and only 27 false positives, amounting to a precision of 99.00%. In the prime case, the detection strategy consisted of looking for the first ladder iteration by locating in the traces for a cache-hit matching the execution of ec_GFp_simple_dbl()and then counting the number of consecutive cache hits matching the execution of BN_copy(). If the next two slots had cache hits for the latter, this means that the copy took around 3 slots or 15,000 cycles. When running the attack against 10,000 signature computations, 2.343 signatures with second MSB 0 and only 12 false positives, amounting to a precision of 99.53%.

For recreating this exploit, there is a GitHub repository that contains the attack code to exploit small side-channel leakage.

Impact

In December 2019, vulnerabilities in versions 1.0.2t and 1.0.1l in accordance with the OpenSSL security policy before the end of long-term support for those versions were originally reported to the OpenSSL development team. After version 1.0.2u was released, it was confirmed that the same vulnerabilities were still present, and hence a patch was proposed with corresponding countermeasures, which has already been approved. Although the 1.0.2 branch is now out of public support as of May 2020, the OpenSSL development team still provides its customers with extended support for 1.0.2, and the patch is included in their further internal releases. While it is hard to estimate how the vulnerability would affect real products due to the lack of public releases containing the patch, searching over GitHub revealed several projects updating their build process to take into account a new release from the 1.0.2 branch. Similar steps have also been taken to address the vulnerability in RELIC, and a fix was pushed in January 2020.

Mitigation

- The countermeasure that is easiest to implement randomization of coordinates to guarantee all intermediate points in project coordinates.
- Another potential countermeasure is to refactor the implementation to satisfy the constant-time guarantee.

References

1. https://eprint.iacr.org/2020/615.pdf
2. https://github.com/akiratk0355/ladderleak-attack-ecdsa

6.2 Certificates related flaws

6.2.1 GitHub homepage down due to expired certificate

Introduction

GitHub, Inc. is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features.

Digital certificates, also known as identity certificates or public key certificates, are digital files that are used to certify the ownership of a public key. SSL certificates are a type of digital certificate, issued by a Certificate Authority (CA). The CA signs the certificate, certifying that they have verified that it belongs to the owners of the domain name which is the subject of the certificate.
SSL certificates usually contain the following information:

- The subject domain name
- The subject organization
- The name of the issuing CA
- Additional or alternative subject domain names, including subdomains, if any
- Issue date
- Expiry date
- The public key (The private key, however, is a secret.)
- The digital signature by the CA

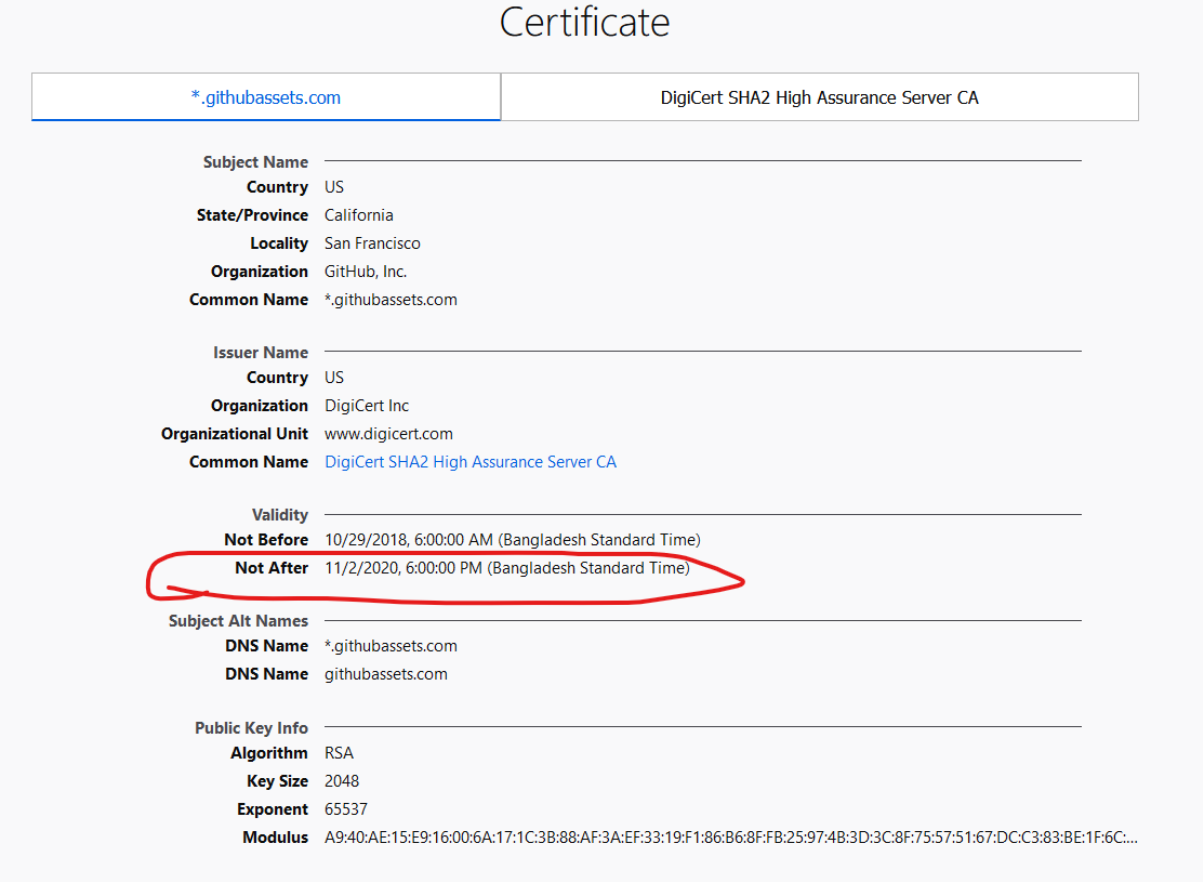
When a user tries to connect to a server, the server sends them its SSL certificate. The user then verifies the server's certificate using CA certificates that are present on the user's device to establish a secure connection. This verification process uses public-key cryptography, such as RSA or ECC, to prove the CA signed the certificate. As long as you trust the CA, this demonstrates you are communicating with the server certificate's subject (e.g., ProtonMail.com).

CVSS Score: 5 / Medium

Description

GitHub's home page malfunctioned, with reports indicating that an expired SSL certificate was to blame.

GitHub users reported being unable to access various resources, apparently because the SSL certificate issued to GitHub's content delivery network (CDN) was only valid until November 2, 2020, 7:00 AM ET.



Content Delivery Networks (CDNs) comprise distributed sets of servers, separate from the main website's server which are strategically placed at different geographical locations.

This is done to optimize the performance, speed, and delivery of content like videos, images, and other web resources.

For example, while the main *github.com* server may be hosting the text you can read on the website, the images, stylesheets, and JavaScript files may be coming from a completely separate CDN server, depending on your location and other factors.

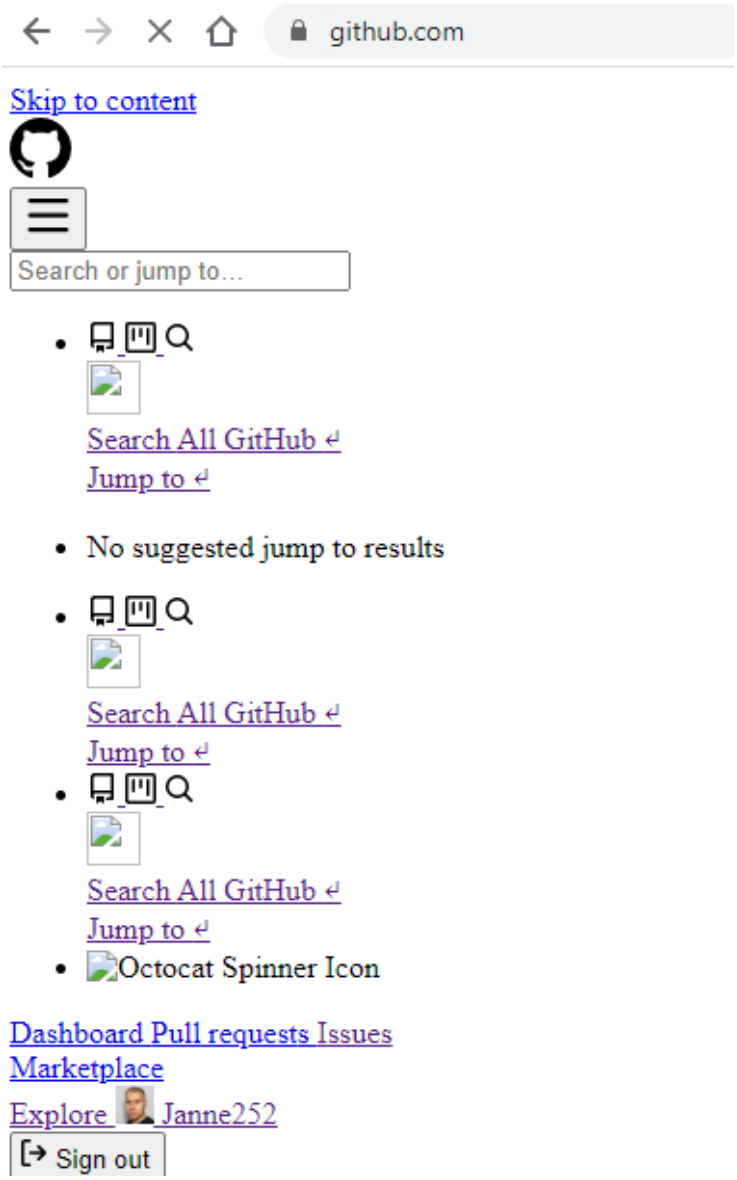
Exploitation

Since *https*://github.com is hosted on a secure server with a valid SSL certificate, the website would *not* automatically pull images from a CDN with an expired SSL certificate, without throwing warnings, or in some cases breaking the website's UI altogether. This is called the ***mixed content problem***.

Mixed content occurs when initial HTML is loaded over a secure HTTPS connection, but other resources (such as images, videos, stylesheets, scripts) are loaded over an insecure HTTP connection. This is called mixed content because both HTTP and

HTTPS content are being loaded to display the same page, and the initial request was secure over HTTPS.

A Software Developer tweeted to the company too about this problem with the following screenshot:



Impact

- *github.com* would show text, links, and thumbnails fine, but was devoid of its rich UI, stylesheets, and scripts that make the open-source repository look whole.
- GitHub had experienced a downtime of about **30 minutes**
- Because of the mixed content problem, this would have been the impact in case GitHub didn't act quickly**:**
    - By requesting subresources using the insecure HTTP protocol weakens the security of the entire page, as these requests are vulnerable to on-

path attacks, where an attacker eavesdrops on a network connection and views or modifies the communication between two parties.

Mitigation

- A new certificate has been installed on the *github.githubassets.com* domain to remediate the issue.
  - This new certificate will, however, expire in November 2021.
- Someone on GitHub should remind someone now to renew their SSL certificate before it expires this year.
- The Mixed Content Problem can be fixed by the following approaches:
  - Automatically insert the upgrade-insecure-requests CSP (Content Security Policy) directive
  - Modify all links to use https://
  - Create a system that rewrites known HTTPS capable URIs

References

1. https://www.techradar.com/news/github-home-page-down-after-apparent-ssl-fail
2. https://www.bleepingcomputer.com/news/security/github-breaks-site-layout-after-forgetting-to-renew-certificate/
3. https://blog.cloudflare.com/fixing-the-mixed-content-problem-with-automatic-https-rewrites/

6.2.2 CVE-2020-0601: Flaw in Windows CryptoAPI (Crypt32.dll) implementation of Elliptic Curve Cryptography (ECC) can result in certificate forgery

Introduction

A spoofing vulnerability exists in the way Windows CryptoAPI (Crypt32.dll) validates Elliptic Curve Cryptography (ECC) certificates.

An attacker could exploit the vulnerability by using a spoofed code-signing certificate to sign a malicious executable, making it appear the file was from a trusted legitimate source. The user would have no way of knowing the file was malicious because the digital signature would appear to be from a trusted provider.

A successful exploit could also allow the attacker to conduct man-in-the-middle attacks and decrypt confidential information on user connections to the affected software.

CVSS Score: 8.1 / High

Description

A spoofing vulnerability exists in the way Windows CryptoAPI (Crypt32.dll) validates Elliptic Curve Cryptography (ECC) certificates.A successful exploit could also allow the attacker to conduct man-in-the-middle attacks and decrypt confidential information on user
connections to the affected software. As a result, an attacker may be able to craft a certificate that appears to have the ability to be traced to a trusted root certificate authority.

Any software, including third-party non-Microsoft software, that relies on the Windows CertGetCertificateChain() function to determine if an X.509 certificate can be traced to a trusted root CA may incorrectly determine the trustworthiness of a certificate chain. Microsoft Windows versions that support certificates with ECC keys that specify parameters are affected.

Exploitation

Code Signing certificates are primarily used to validate the executable files to trust the source and verify it is coming from the legitimate trusted source. Typical use case to perform code signing is via code signing via the private key of the certificate and it is verified using the public key of the certificate.

The Microsoft Windows CryptoAPI, which is provided by Crypt32.dll fails to validate ECC certificates in a way that properly leverages the protections that ECC cryptography should provide. As a result, an attacker may be able to craft a certificate that appears to have the ability to be traced to a trusted root certificate authority.

Any software, including third-party non-Microsoft software, that relies on the Windows CertGetCertificateChain() function to determine if an X.509 certificate can be traced to a trusted root CA may incorrectly determine the trustworthiness of a certificate chain.

Microsoft Windows versions that support certificates with ECC keys that specify parameters are affected. This includes Windows 10 as well as Windows Server 2016 and 2019. Windows 8.1 and prior, as well as the Server 2012 R2 and prior counterparts does not support ECC keys with parameters. For this reason, such certificates that attempt to exploit this vulnerability are inherently untrusted by older Windows versions.

Impact

By exploiting this vulnerability, an attacker may be able to spoof a valid X.509 certificate chain on a vulnerable Windows system. This may allow various actions including, but

not limited to, interception and modification of TLS-encrypted communications or spoofing an Authenticode signature.

Mitigation

This vulnerability is addressed in the Microsoft Update for **CVE-2020-0601**

References

1. https://research.kudelskisecurity.com/2020/01/15/cve-2020-0601-the-chainoffools-attack-explained-with-poc/

6.2.3 Certificate Expiration Takes Down Spotify

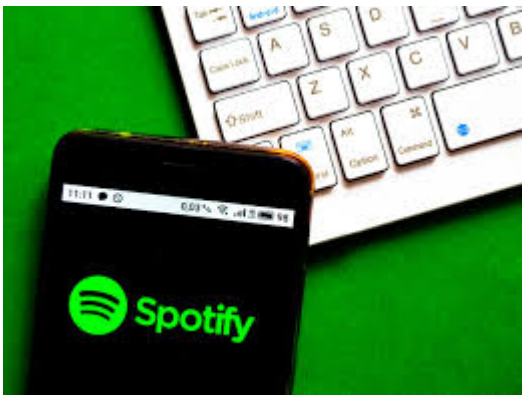
Introduction

Spotify, a Swedish audio streaming and media service provider services were down due to expired certificates. Spotify customers spread across the world faced this issue.

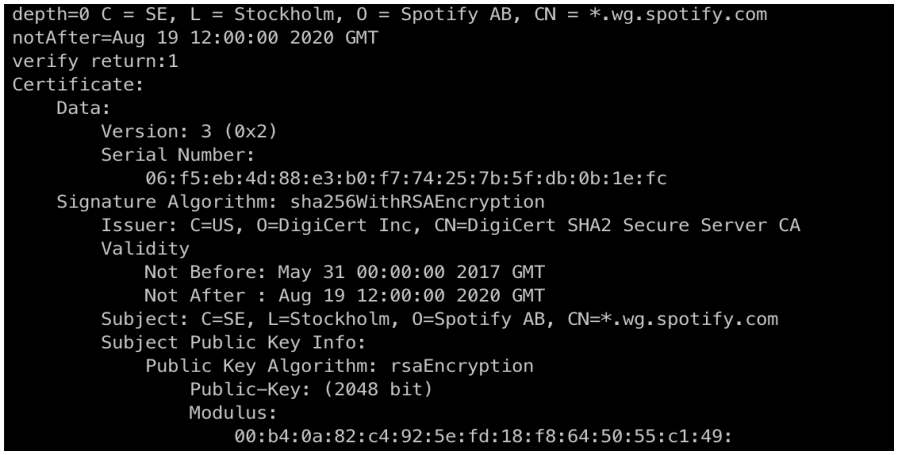
CVSS Score: 5.3 / Medium (severity for an expired certificate is reported as per NASL/Tenable standard)

Description

Spotify customers worldwide were not able to access the Spotify streaming service on 19th of Aug 2020.



The outage was addressed to the expired certificates and the same was evident by the screenshots of Spotify Certificate details shared by various twitter users.



```
depth=0 C = SE, L = Stockholm, O = Spotify AB, CN = *.wg.spotify.com
notAfter=Aug 19 12:00:00 2020 GMT
verify return:1
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            06:f5:eb:4d:88:e3:b0:f7:74:25:7b:5f:db:0b:1e:fc
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=DigiCert Inc, CN=DigiCert SHA2 Secure Server CA
        Validity
            Not Before: May 31 00:00:00 2017 GMT
            Not After : Aug 19 12:00:00 2020 GMT
        Subject: C=SE, L=Stockholm, O=Spotify AB, CN=*.wg.spotify.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:b4:0a:82:c4:92:5e:fd:18:f8:64:50:55:c1:49:
```

Source : Tweet by @lpoinsig

Exploitation

A service provider like Spotify needs no external adversary to harm them in case of certificate expiration. As incidents like these are not triggered by external threat actors, it has very negligible chances of getting exploited.

Impact

There certainly is some reputational damage but financial implications are higher for the companies like Spotify whose revenue is dependent on the availability of their service to their customers.

The theoretical impact to Spotify around **$1.27 M** for the one and a half hour downtime. Since the Spotify revenue comes for subscriptions, the actual impact is nearly zero in dollars, but for other businesses the impact can be different.

Here are some common impacts of certificate expiration to different businesses

- Retail Sectors (Amazon, Ebay) - Direct revenue losses
- Travel bookings - directly affects customer retention & acquisition
- Authentication services - Increased customer support time/costs
- Cloud infrastructure providers - Customer retention
- Damage to brand reputation & compliance posture

Mitigation

It takes changes at the operational level to monitor expiring certificates and renew them proactively. You can look for Null - Cryptography Club tools/services to send you advance warnings and reminders to renew your expiring certificates.

Procuring long validity certificates is not a potential mitigation as it may create other security issues and operation overheads in case an organisation has to revoke the certificate for any reason.

Reference

- https://www.thesslstore.com/blog/the-day-the-music-died-certificate-expiration-takes-down-spotify/

6.3 Buffer overflow

6.3.1 "pool-based" buffer overflow vulnerability in the Windows Kernel Cryptography Driver

Introduction

CVE-2020-17087 is a pool-based buffer overflow vulnerability in the Windows Kernel Cryptography Driver (cng.sys). The vulnerability arises from input/output controller (IOCTL) 0x390400 processing and could allow a local attacker to escalate privileges, including for sandbox escape. The vulnerability was initially released as a zero-day by Google's Project Zero team; it was patched on November 10, 2020, as part of Microsoft's November 2020 Patch Tuesday release.

The vulnerability was identified in the processing of IOCTL 0x390400, reachable through the following series of calls:

1. cng!CngDispatch
2. cng!CngDeviceControl
3. cng!ConfigIoHandler_Safeguarded
4. cng!ConfigFunctionIoHandler
5. cng!_ConfigurationFunctionIoHandler
6. cng!BCryptSetContextFunctionProperty
7. cng!CfgAdtReportFunctionPropertyOperation
8. cng!CfgAdtpFormatPropertyBlock

CVSS Score: 7.8 / High

Description

Pool buffer overflow due to an incorrect 16-bit integer truncation in the processing of IOCTL 0x390400. The bug is in the function cng!CfgAdtpFormatPropertyBlock. To calculate the size of the buffer to allocate, the function multiplies an attacker-controlled integer by 6 and stores the result in a variable of type USHORT. If the input buffer length passed in during the IOCTL call is greater than or equal to 0x2AAB, then the buffer allocated by BCryptAlloc() will be too small, leading to a buffer overflow.

The flaw can be exploited by attackers for arbitrary code execution by getting the targeted user to access a website hosting a specially crafted font file.

Chaining the Windows and Chrome vulnerabilities, the attackers can escape the Chrome sandbox and execute malicious code on the targeted system.

Exploitation

The exploit uses the buffer overflow to establish an arbitrary read/write primitive in the kernel address space with the help of Named Pipe objects.

The bug resides in the cng!CfgAdtpFormatPropertyBlock function and is caused by a 16-bit integer truncation issue. It is best explained with a C-like pseudo code of the function:

--- cut ---

```
1: NTSTATUS CfgAdtpFormatPropertyBlock(PBYTE SourceBuffer, USHORT
SourceLength, PUNICODE_STRING Destination) {
2:      CONST USHORT DestinationSize = (USHORT)(6 * SourceLength);
3:      PWCHAR OutputBuffer = BCryptAlloc(DestinationSize);
4:
5:      for (USHORT i = 0; i < SourceLength; i++) {
6:              *OutputBuffer++ = "0123456789abcdef"[*SourceBuffer >> 4];
7:              *OutputBuffer++ = "0123456789abcdef"[*SourceBuffer & 0xF];
8:              *OutputBuffer++ = ' ';
9:              SourceBuffer++;
10:     }
11:
12:     Destination->MaximumLength = DestinationSize;
13:     Destination->Length = DestinationSize - 2;
14:     Destination->Buffer = OutputBuffer;
15:
```

```
16:     return STATUS_SUCCESS;

17: }

--- cut ---
```

The integer overflow occurs in line 2, and if SourceLength is equal to or greater than 0x2AAB, an inadequately small buffer is allocated from the NonPagedPool in line 3. It is subsequently overflown by the binary-to-hex conversion loop in lines 5-10 by a multiple of 65536 bytes.

Impact

There is a total loss of confidentiality, resulting in all resources within the impacted component being divulged to the attacker. Alternatively, access to only some restricted information is obtained, but the disclosed information presents a direct, serious impact.

There is a total loss of integrity or a complete loss of protection. For example, the attacker is able to modify any/all files protected by the impacted component. Alternatively, only some files can be modified, but malicious modification would present a direct, serious consequence to the impacted component.

There is a total loss of availability, resulting in the attacker being able to fully deny access to resources in the impacted component; this loss is either sustained (while the attacker continues to deliver the attack) or persistent (the condition persists even after the attack has completed). Alternatively, the attacker has the ability to deny some availability, but the loss of availability presents a direct, serious consequence to the impacted component (e.g., the attacker cannot disrupt existing connections, but can prevent new connections; the attacker can repeatedly exploit a vulnerability that, in each instance of a successful attack, leaks an only small amount of memory, but after repeated exploitation causes a service to become completely unavailable).

Mitigation

- Use static analysis to identify places in the kernel code where inadequately small integer types are used to store buffer lengths, and refactor them or discourage their use in the future.
- Potential further hardening of the kernel pool allocator to make the exploitation of memory corruption in the Windows kernel (especially large overflows that corrupt more than 64 kB) less reliable.

SanerNow offers the *detection* and *remediation* for CVE-2020-15999. It can also *detect* the affected Windows OS for CVE-2020-17087. Patch for the same is currently available from Microsoft.

References

1. https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2020-17087
2. https://www.helpnetsecurity.com/2020/11/02/cve-2020-17087/
3. https://bugs.chromium.org/p/project-zero/issues/detail?id=2104
4. https://googleprojectzero.github.io/0days-in-the-wild/0day-RCAs/2020/CVE-2020-17087.html

6.3.2 CVE-2020-36242 - Python crypto lib

Introduction

Integer overflow also known as wraparound, occurs when an arithmetic operation outputs a numeric value that falls outside allocated memory space or overflows the range of the given value of the integer. Mostly in all programming languages, integer values are allocated limited bits of storage.

Most compilers will ignore the overflow and store unexpected output or errors. This will result in various attacks such as buffer overflow which is the most common attack and leads to executing malicious programs or privilege escalation.

CVSS Score: 9.1 / Critical

Description

In the cryptography package before 3.3.2 for Python, certain sequences of update() call to symmetrically encrypt multi-GB values that could result in an integer overflow and buffer overflow, as demonstrated by the Fernet class. This leads to a buffer overflow in OpenSSL as well.

Using CWE to declare the problem leads to CWE-190. Impacted is confidentiality, integrity, and availability

Exploitation

- The attack needs to approach within the local network. The successful exploitation requires a simple authentication.
- The data_size must satisfy the following condition

plaintext != original_plaintext if _CipherContext._MAX_CHUNK_SIZE is set to 2 ** 31 - 1 - 8

```
import cryptography.fernet as cf

data_size = 2**32 - 1
```

```
original_plaintext = bytes(data_size)

fernet = cf.Fernet(cf.Fernet.generate_key())

ciphertext = fernet.encrypt(original_plaintext)

plaintext = fernet.decrypt(ciphertext)

assert plaintext == original_plaintext
```

- Fails with SIGABRT or SIGSEGV and stderr: "free(): invalid size" or "munmap_chunk(): invalid pointer" or cryptography.exceptions.InternalError: Unknown OpenSSL error.
- This error is commonly encountered when another library is not cleaning up the OpenSSL error stack.

([_OpenSSLErrorWithText(code=101560482, lib=6, func=219, reason=162, reason_text=b'error:060DB0A2:digital envelope routines:evp_EncryptDecryptUpdate:partially overlapping buffers')])

Impact

- All the modules that use cryptography package before 3.3.2 are affected
- Caused buffer overflow in openssl
- Successful exploitation of overflow vulnerabilities can lead to remote code execution at the best and denial of service at the least

Mitigation

- Update the cryptography package version that is used
- If the data size is greater than _CipherContext._MAX_CHUNK_SIZE = 2 ** 31 - 1 - 16, make sure that your program splits the data and encrypts/decrypts to avoid this issue.
- Secure coding practices must be implemented

References

1. https://github.com/advisories/GHSA-rhm9-p9w5-fwm7
2. https://github.com/alex/cryptography/commit/fae28c1a1c82a8a5a0ddd0d47c441d5781cca45d
3. https://nvd.nist.gov/vuln/detail/CVE-2020-36242

6.4 Certifying Authority Best Practices gap

6.4.1 Let's Encrypt has to revoke 3 Million certificates due to CAA Rechecking bug

Introduction

Let’s Encrypt – the world-leading Free SSL Certificate authority (CA), has announced that it will revoke more than 3 million SSL/TLS certificates by 4th March 2020. The cause of the revocation is a bug that was discovered by Let’s Encrypt.

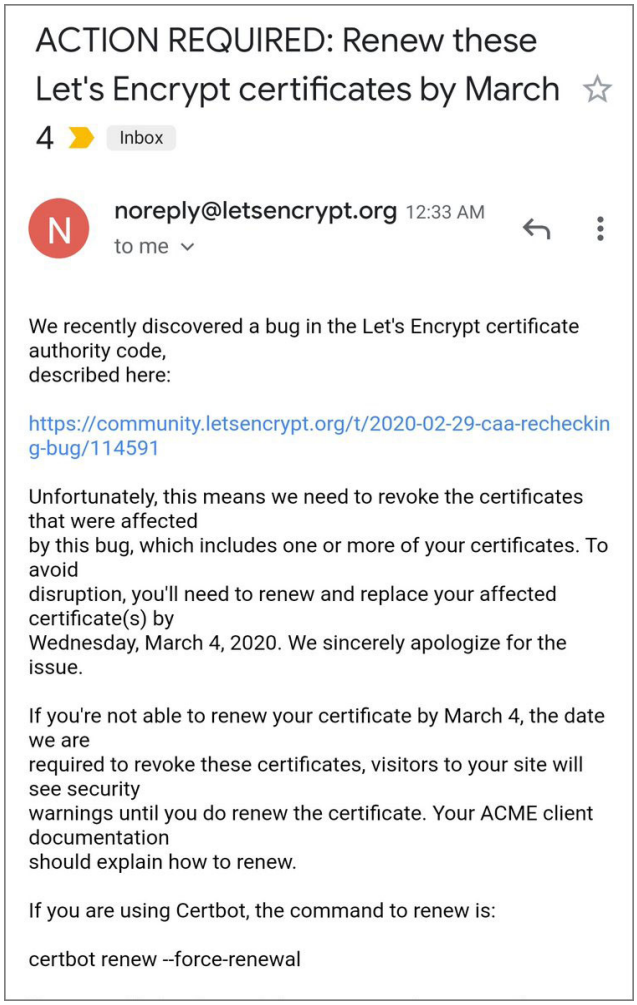
Let’s Encrypt confirmed that a bug in Boulder ignored CAA Checks in a forum post on 29th February 2020. However, this news barely gave time to their users to react to it.

CVSS Score: 6.4 / Medium

Description

Let’s Encrypt announced there was a bug in their code that allowed the issuance of SSL Certificates without going through proper domain record checks. This resulted in Encrypt revoking more than 3 million valid SSL certificates out of their total 116 million certificates. To be more specific, the bug affected Boulder – the server software that Let’s Encrypt uses to verify the users and their domains before issuing an SSL certificate.

For those affected, Let's Encrypt had emailed users who must renew their certificates by March 4, 2020, before they can become invalid.



ACTION REQUIRED: Renew these Let's Encrypt certificates by March 4

Inbox

noreply@letsencrypt.org 12:33 AM
to me

We recently discovered a bug in the Let's Encrypt certificate authority code,
described here:

https://community.letsencrypt.org/t/2020-02-29-caa-rechecking-bug/114591

Unfortunately, this means we need to revoke the certificates that were affected
by this bug, which includes one or more of your certificates. To avoid
disruption, you'll need to renew and replace your affected certificate(s) by
Wednesday, March 4, 2020. We sincerely apologize for the issue.

If you're not able to renew your certificate by March 4, the date we are
required to revoke these certificates, visitors to your site will see security
warnings until you do renew the certificate. Your ACME client documentation
should explain how to renew.

If you are using Certbot, the command to renew is:

certbot renew --force-renewal

Exploitation

When a certificate request contained N domain names that needed CAA rechecking, Boulder (The CA Software) would pick one domain name and check it N times. What this means in practice is that if a subscriber validated a domain name at time X, and the CAA records for that domain at time X allowed Let's Encrypt issuance, that subscriber would be able to issue a certificate containing that domain name until X+30 days, even if someone later installed CAA records on that domain name that prohibit /issuance by Let's Encrypt.

Impact

This bug caused a domain on a multi-domain certificate to be checked numerous times rather than all the domains on the certificate being checked once. This caused certificates to be issued without the proper CAA checks for some domains.

Mitigation

The CA has released an online tool to help subscribers determine whether they use affected certificates and also published a list of affected serial numbers.

Once you have determined that you are using the impacted Let's Encrypt certificate, The only way to mitigate this is to renew the certificate. Users can renew the certificate either from a Trusted Certificate Authority or go for a Free untrusted SSL Certificate Authority.

Reference

- https://www.ssl.support/blog/lets-encrypt-certificates-been-revoked/
- https://www.zdnet.com/article/lets-encrypt-to-revoke-3-million-certificates-on-march-4-due-to-bug/
- https://www.bleepingcomputer.com/news/security/lets-encrypt-to-revoke-3-million-tls-certificates-due-to-bug/

6.4.2 Violation of the Baseline Requirements by Microsoft's intermediate CA platform while generating OCSP responder certificate

Introduction

The OCSP protocol is used to validate the revocation status of the certificate without requesting the CA in person. The given issue is a violation of the baseline requirements of rfc6960, which states that any intermediate CA cannot sign OCSP responder certificate unless the intermediate certificate does have EKU having OCSP signing (id-kp-OCSPSigning) attribute present or another certificate itself needs to be issued having same public key and subjectDN as the Intermediate CA in problem , but signed with any untrusted CA would result in desired outcome of signing OCSP responder certificate. However, this is against the policy described and henceforth is the vulnerability or the issue from Microsoft CA platform.

CVSS Score: Medium

Description

In compliance with **rfc6960**, Online Certificate Status Protocol that is OCSP is used to validate the revocation status of the certificate without requesting Certificate Authority in person and is an automated response by the server. In the process of OCSP response, the intermediate certificate or any certificate derived from root CA would not directly sign the OCSP response, instead of that it signs an OCSP responder certificate that would be used to sign OCSP responses.

So the process of OCSP response would be , any intermediate certificate was to first sign OCSP responder certificate, and the OCSP responder certificate was to include an EKU with id-kp-OCSPSigning.

Vulnerability

This is a compliance issue only with respect to Microsoft CA platform.

Microsoft CA platform requires EKU(Extended Key Usage) chaining when it comes to OCSP response .

In simple words, any intermediate CA cannot sign OCSP responder certificate unless the intermediate certificate does have EKU having OCSP signing (id-kp-OCSPSigning) attribute present or another certificate itself needs to be issued having same public key and subjectDN as the Intermediate CA in problem , but signed with any untrusted CA would result in desired outcome of signing OCSP responder certificate. However, this is against the policy described and henceforth is the vulnerability or the issue from Microsoft CA platform.

Impact

The grant to Microsoft to provide OCSP responses for any certificate provided by root CA to Microsoft is dangerous as Microsoft can sign any OCSP response for any certificate issued by Root CA.

For example,

The certificates issued by DigiCert to Microsoft , grants Microsoft the ability to provide OCSP responses for any certificate issued by Digicert .

Mitigation

Affected CA should include id-pkix-ocsp-nocheck extension to the certificates not to misue these certificates.

Reference

- https://groups.google.com/g/mozilla.dev.security.policy/c/XQd3rNF4yOo/m/bXYjt1mZAwAJ

6.5 Other Categories

6.5.1 'Zerologon' Vulnerability in Netlogon Could Allow Attackers to Hijack Windows Domain Controller

Introduction

Cryptographic flaw in Netlogon, a windows authentication service, allows an attacker in a local network to take control of a Windows domain controller (DC).

CVSS Score: 10 / Critical

Description

Background

Netlogon is a Windows Server process that authenticates users and other services within a domain. It runs as an RPC service with the name Netlogon Remote Protocol (MS-NRPC).

The NRPC uses a call ComputeNetlogonCredential to process user/client credentials as part of NETLOGON_AUTHENTICATOR structure

```
SET TimeNow = current time;
SET ClientAuthenticator.Timestamp = TimeNow;
SET ClientStoredCredential = ClientStoredCredential + TimeNow;
CALL ComputeNetlogonCredential(ClientStoredCredential,
              Session-Key, ClientAuthenticator.Credential);
```

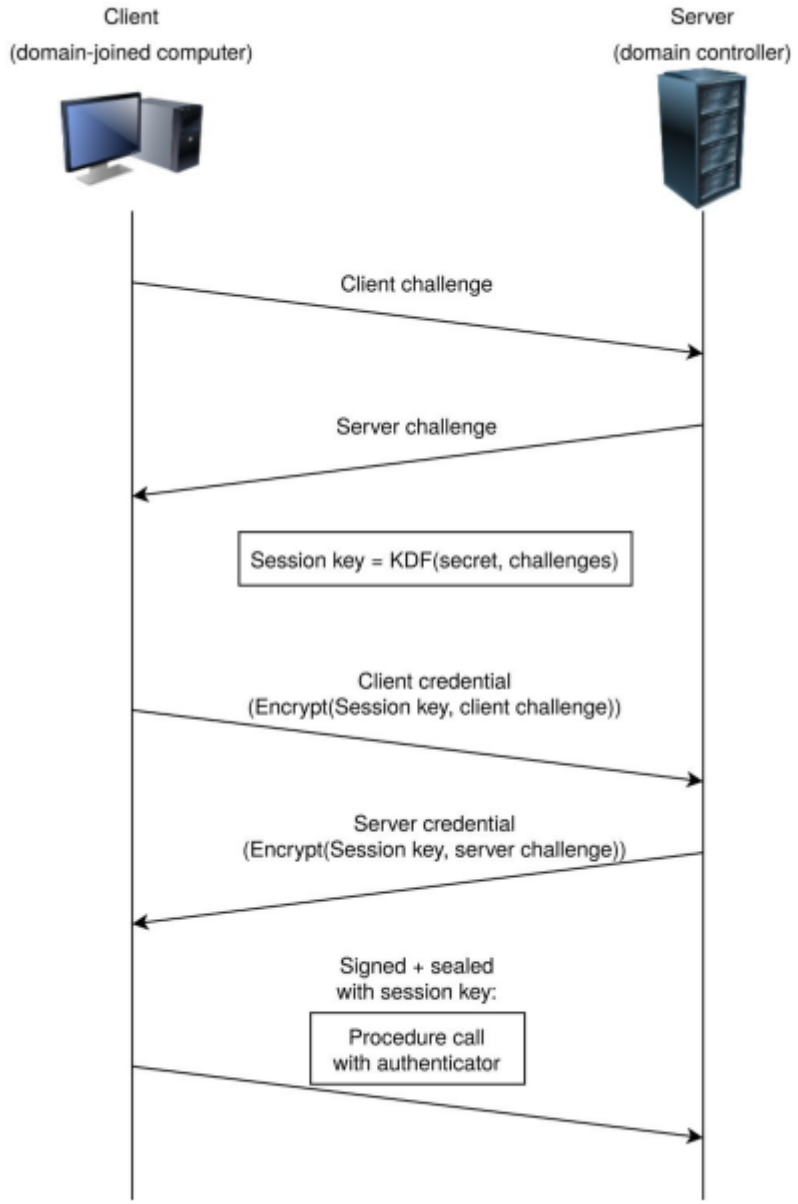
Source - Microsoft Documentation

In the initial NetLogon handshake between any domain joined machine and domain controller, client sends **ClientChallenge** and server sends **ServerChallenge**.

The **ComputeNetlogonCredential** as shown above takes 8 bytes “Session-Key " as a parameter, which is obtained from a key-derivation-function which takes user password and combination of **ClientChallenge** & **ServerChallenge**.

Session-Key = KDF (password, ClientChallenge + ServerChallenge)

An attacker cannot generate Session-Key, unless the attacker is aware of the password.



NetLogon Authentication flow (source)

Next, **ClientChallenge** is encrypted with this **Session-Key** using AES in cipher feedback mode with an 8 bit width (**AES-CFB8**) to obtain **ClientCredential**.

ClientCredential is sent to the server. If the server can compute and match the **ClientCredential** on its side, then the client is successfully authenticated and the server proceeds further with the session.

The flaw

Tom Tervoort, a security specialist from the Secura, in Aug 2020 disclosed multiple issues with the way NetLogon service handels user credentials in this white paper.

1. **Fixed IV**

Windows implementation of ComputeNetLogonCredential function uses fixed IV of 16 byte length with all bytes set to zero. This makes the AES-CFB8 implementation insecure

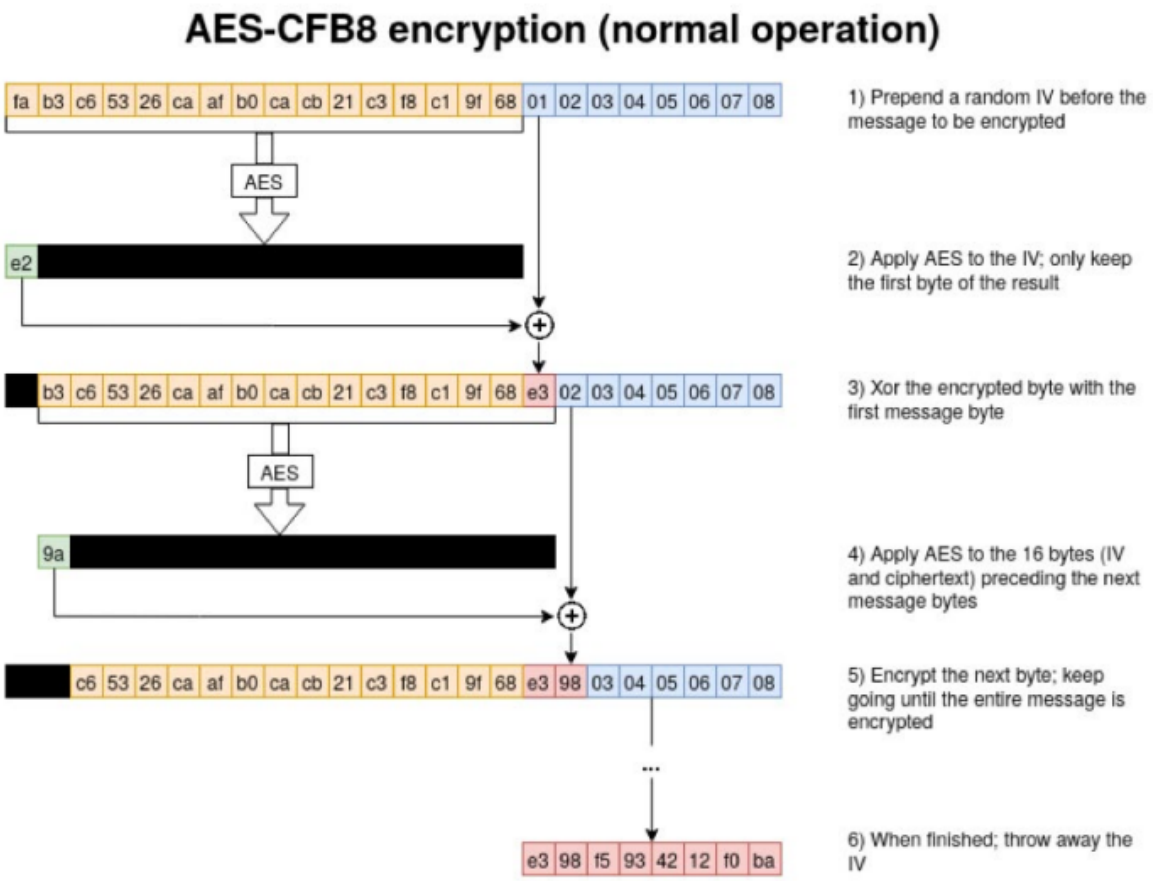
2. **Uncapped authentication attempts**

With no restrictions or lockouts on the number of unsuccessful attempts made by a user, it becomes easy to attempt and spoof user credentials with multiple combinations.

Combining the above flaws can allow an attacker to completely compromise the authentication, and allows an attacker in a local network to take complete control of a Windows domain controller (DC).

3. **All Zero Ciphertext with AES-CFB8**

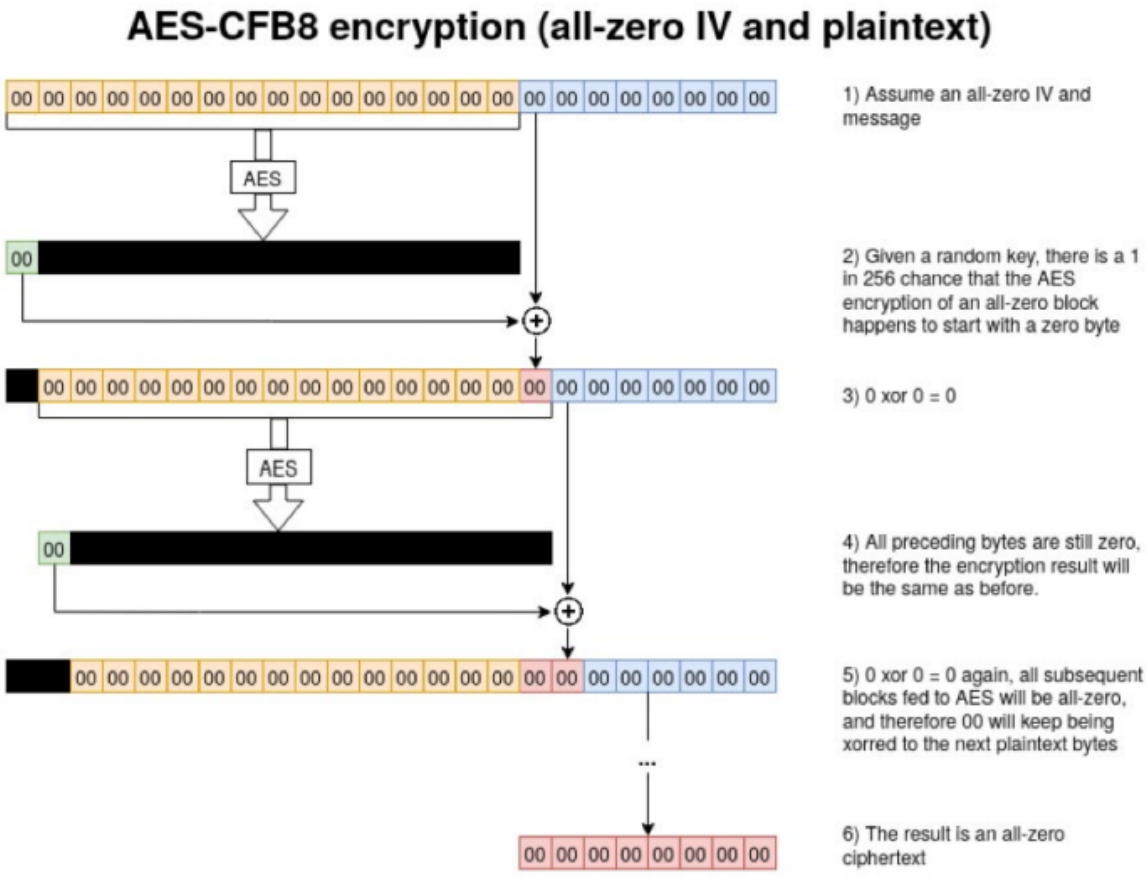
AES-CFB8 encrypts each byte of the plaintext by prepending a 16-byte ‘Initialisation Vector’ to the plaintext, then applying AES to the first 16 bytes of the IV+plaintext, taking the first byte of the AES output, and XOR’ing it with the next plaintext byte.



Normal case of AES-CFB8 Encryption (source)

If an attacker can fix the IV to all-zero bytes, and the **ClientChallenge** is also chosen by the user, then the first byte of Session-Key can be chosen so that the resultant value of ciphertext byte is 0.

Since the attacker is not aware of the password, the attacker cannot successfully generate the **Session-Key** value. By enumeration an attacker can choose a right value within 256 attempts which matches with a valid **Session-Key**.



Encrypting all-zero IV with all-zero message with AES-CFB8 (source)

After successfully guessing the first value, the remaining bytes of session key has less significance and results in a all-zero verifiable **ClientCredential** resulting in successful authentication of the attacker without being aware of the actual password.

Exploitation

With a fixed IV to all-zero bytes and a chosen all-zero byte Client-Challenge, a user has to just enumerate the first byte of 8 byte **Session-Key** to successfully generate an all-zero **ClientCredential** which will be acceptable by server. This can be achieved by enumerating the first byte of a **Session-Key** in just 256 attempts.

With this method a user can log in to any computer in the domain. This includes backup domain controllers, and even the targeted domain controller itself.

Further to this, an attacker can use the authenticated session to set blank passwords or chosen passwords to carry out further attacks.

Impact

Using the Zerologon vulnerability a local user can do following

1. Take the entire control of the domain controller in an organisation
2. Launch a DOS attack on all the users and machines by resetting their credentials
3. Impersonate any and all the users enrolled with AD, know, set and reset their credentials

Mitigation

As far as NetLogon service is concerned, Microsoft have released patches for CVE-2019-1424 & CVE-2020-1472, applying them will fix the issue.

For the cryptography best practices it is advised to avoid use of AES in cipher feedback mode.

Reference

1. https://www.secura.com/uploads/whitepapers/Zerologon.pdf
2. https://nvd.nist.gov/vuln/detail/cve-2020-1472
3. https://www.crowdstrike.com/blog/cve-2020-1472-zerologon-security-advisory/

6.5.2 Identrust OCSP verification was failing

Introduction

While establishing a secure connection using the standard SSL mechanism, we trust digital certificates. The way we can trust them is, for example, if a vendor is Microsoft or Red Hat or Mozilla with the Firefox web browser, has a list of certificate authorities (CA) that it trusts, and provides a bundle of these certificates with the software. So when we surf to our online bank and receive a certificate that is signed by a listed CA, the web browser can check that the signature matches.

From time to time, a certificate might be revoked. Reasons for this could span from security breaches to simply organizational changes. But a certificate that is already signed and delivered to us, is still valid until we are told about the revocation. And there is no automation here. So there is a need for a mechanism where we can ask the CA if the certificate is still valid. This is what OCSP does. The CA provides a web service where a user (like a web browser) can look up a given certificate, and check that it is not revoked. The response is signed by the CA, and is valid for a few days, to avoid hammering the service for each and every request.

CVSS Score: 6.6 / High

Description

IdenTrust experienced a significant increase in requests to the OCSP responder, peaking at 650k requests per second (compared with a typical rate of 10k rps to this responder) and the exact cause of that spike in traffic is unknown. This responder is fronted by a CDN, but these requests almost all hit the origin. As a result, the ISRG OCSP responder started serving errors, and the firewall at the primary data center serving this responder became unresponsive. This was confirmed later in logging of the degradation in the logging event. When monitoring alerted IdenTrust of potential system impact at 05:23 on 2020-04-24, IdenTrust began troubleshooting to try to identify a cause and if there was impact and the issue appeared to have cleared as traffic returned to normal levels.

Later on 2020-04-24 the issue reoccurred and system engineers again were troubleshooting the cause of the increase in traffic. IdenTrust eventually performed a hard boot on the firewall at the primary data center which required IdenTrust staff to travel to the site. After the firewall was restarted, the request rate dropped back to normal levels and the OCSP responder stopped serving errors. The outage was lengthened by the hard boot that required air travel to the datacenter; during this time, users of this OCSP responder were receiving errors.
As the impact was limited to this particular OCSP responder and it was believed to have only been spikes in traffic that stressed the infrastructure but they didn't see signs of impact at that time as they believed at the time that traffic successfully failed over to the secondary site, we considered that this incident didn't require further actions. As they became aware on 2020-05-08 through the community that there was a report of impact and they have been re-evaluating the incident.

Exploitation

The OCSP server of Identrust was down in April 2020. Usually few clients do OCSP checks of the intermediate certificate, thus this probably doesn't show up very often. It was noticed by some user because he/she had some monitoring set up using gnutls-cli with OCSP checks to verify if certificates are okay. The person checked with:

gnutls-cli --ocsp letsencrypt.org:443 2

This is what the response was:

Connecting to OCSP server: isrg.trustid.ocsp.identrust.com 10…

Resolving 'isrg.trustid.ocsp.identrust.com:80'…

Connecting to '2a02:26f0:3100::1735:2a09:80'…

importing response: ASN1 parser: Error in TAG.

Another user also checked regarding this issue and this is what he got in the response:

```
$ openssl ocsp -no_nonce -url "http://isrg.trustid.ocsp.identrust.com" \

-issuer dst.pem -cert chain.pem  -text

OCSP Request Data:

   Version: 1 (0x0)

    Requestor List:

      Certificate ID:

        Hash Algorithm: sha1

        IssuerName Hash:6FF4684D4312D24862819CC02B3D472C1D8A2FA6

        Issuer Key Hash: C4A7B1A47B2C71FADBE14B9075FFC41560858910

        Serial Number: 0A01414200000153857360A0B85ECA708

Error querying OCSP responder

140671078831424:error:27076072:OCSP routines:parse_http_line1:server
response error:../crypto/ocsp/ocsp_ht.c:260:Code=503,Reason=Service
Unavailable
```

Impact

Privacy compromise. Browser leaks what website is being accessed and who accesses it to CA servers.

Mitigation

The OCSP response checker function now also verifies that the certificate id is the correct one.

Reference

- https://community.letsencrypt.org/t/identrust-ocsp-producing-errors/120677/5
- https://medium.com/@alexeysamoshkin/how-ssl-certificate-revocation-is-broken-in-practice-af3b63b9cb3

6.5.3 TOR nodes stripped TLS off of traffic going to some Bitcoin sites

Introduction

During Aug 2020 nusenu, a researcher keeping track of security and sanity of TOR network, tracked and reported many malicious exit nodes targeting user traffic going to cryptocurrency related websites. He also observed the operation, takedown and recovery of such malicious exit nodes and documented them in his blog.

CVSS Score: 7.5 / High

Description

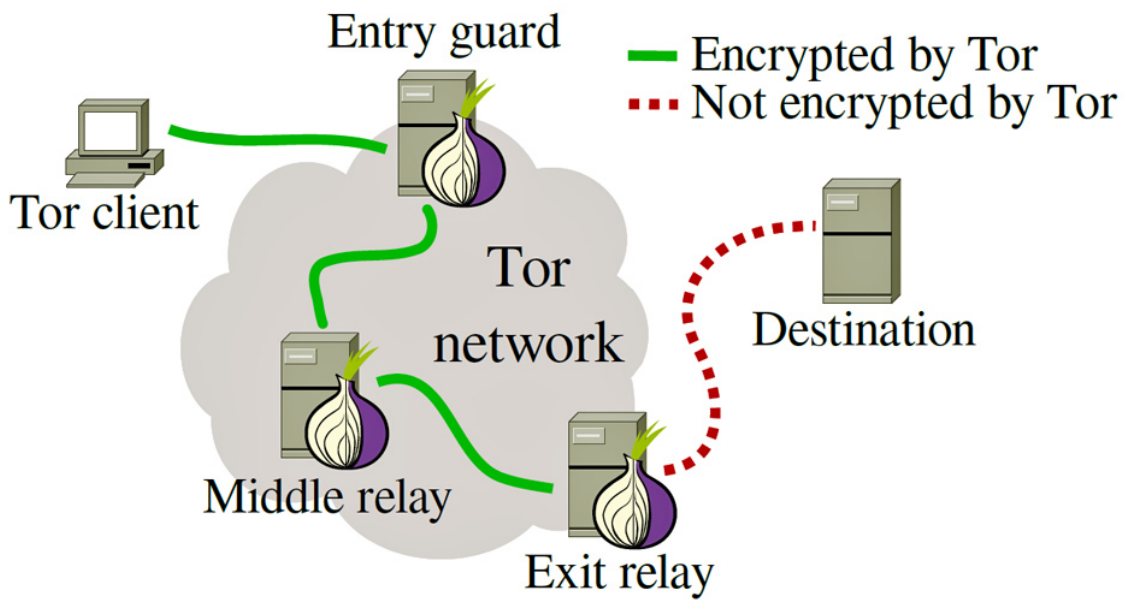
The TOR Network

The Tor network is a Open Source Project which uses end-to-end encrypted communication and a series of volunteer set relay nodes, which are used in series to hide IP address, online data, and browsing history of a user.

The Problem

Many cryptocurrency users use TOR networks to privately transact and protect their identity being traced to any transactions.

Many voluntarily set TOR nodes placed at the last leg of the relay called exit nodes can maliciously force insecure, unencrypted protocol on any transaction using TLS strip attack. The insecure data can be then manipulated to alter the cryptocurrency transaction in favour of the TOR node owner.



nusenu reported malicious actors controlling 23% of the exit nodes were performing "person-in-the-middle attacks by stripping TLS off the HTTPS traffic and forcing HTTPS traffic, specifically targeting users accessing cryptocurrency-related websites using the Tor software or Tor Browser.

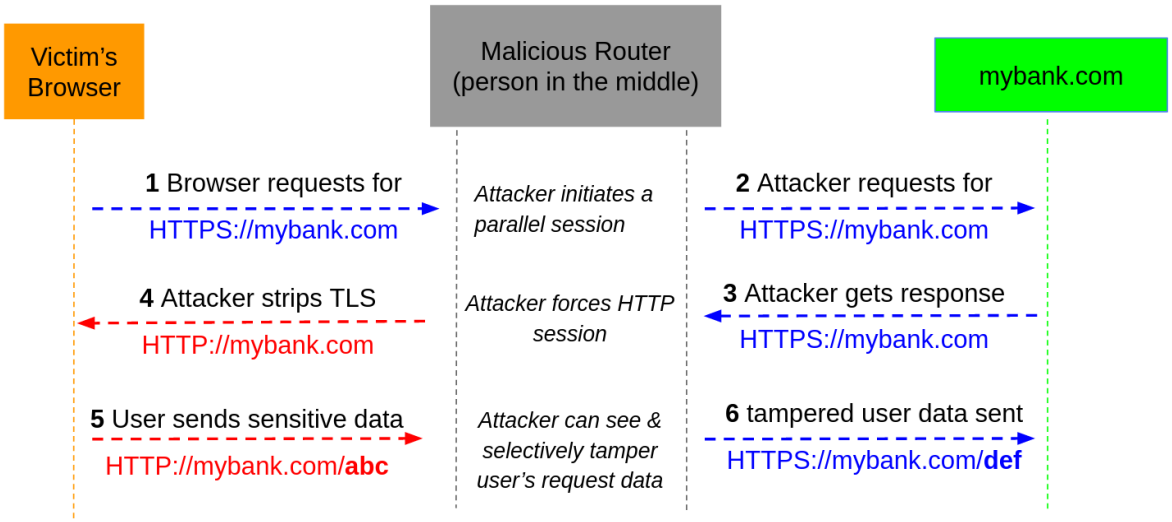
Nusenu said the primary goal of these SSL stripping attacks was to allow the group to replace Bitcoin addresses inside HTTP traffic going to Bitcoin mixing services.

Bitcoin mixers websites are used to send Bitcoin from one address to another by breaking the funds into small sums and transferring them through thousands of intermediary addresses before re-joining the funds at the destination address. By replacing the destination address at the HTTP traffic level, the attackers effectively hijacked the user's funds without the users or the Bitcoin mixer's knowledge.

Exploitation

TLS Stripping is a network based attack discovered by Moxie Marlinspike at Black Hat 2009

This technique leverages the ability of the person in the middle to downgrade user connection from secure HTTPS to insecure HTTP.
Once the connection is downgraded, the person in the middle can inspect and manipulate the data exchanged between user and the server without the user being aware of it or the user's browser giving any warning to the user.



nusenu reported the same TLS Stripping attacks being carried out by 23% of malicious TOR exit nodes. It is hard to detect for Tor Browser users that do not specifically look for the "https://" in the URL bar. This is a well-known attack called "ssl stripping" that exploits the fact that users rarely type in the full domain starting with "https://". There are established countermeasures, namely HSTS Preloading and HTTPS Everywhere, but in practice many website operators do not implement them and leave their users vulnerable to this kind of attack.

- This kind of attack is not specific to Tor Browser. Malicious relays are just used to gain access to user traffic. To make detection harder, the malicious entity did not attack all websites equally. It appears that they are primarily after cryptocurrency-related websites — namely multiple bitcoin mixer services.
- They replaced bitcoin addresses in HTTP traffic to redirect transactions to their wallets instead of the user-provided bitcoin address. Bitcoin address rewriting attacks are not new, but the scale of their operations is.
- It is not possible to determine if they engage in other types of attacks.

Impact

The malicious relays intercept the cryptocurrency transactions submitted to the mining nodes and replaces the receiver's address with their own address to receive all coins involved in the transactions.

This results in financial as well as privacy implications for the victims.

Mitigation

The Tor Project working group will be taking necessary steps to fight this asymmetric and difficult to conquer fight with malicious actors. We will focus on what users and Cryptocurrency exchange/nodes can do to ensure their own safety and security of their transaction.

Suggestions for Service providers

- Implement HSTS (HTTPS Strict Transport Security) tokens to mitigate TLS strip attacks. Learn more about HSTS here.
- Strictly disable all non-TLS communications and ports accessible from internet
- Disable redirection from secure to non-secure communication channels and vice versa.

Suggestion for Endusers

1. Ensure that you are connected to a service provider over a secure channel only ie. HTTP**S** and not HTTP
2. Update browsers regularly and not install untrusted extensions.
3. Change and choose your Tor relays if you get any certificate warnings while connecting to service providers

Reference

- https://www.cloudflare.com/en-in/learning/ssl/transport-layer-security-tls/
- https://www.zdnet.com/article/a-mysterious-group-has-hijacked-tor-exit-nodes-to-perform-ssl-stripping-attacks/
- https://nusenu.medium.com/how-malicious-tor-relays-are-exploiting-users-in-2020-part-i-1097575c0cac

6.5.4 wolfSSL TLS 1.3 Client Man-in-the-Middle Attack CVE-2020-24613

Introduction

wolfSSL is an open-source library with commercial support. It implements the TLS protocol and all the necessary cryptographic functions and is written in the C language, with support for bindings from other languages. The library is supported on a variety of platforms and is widely used in embedded devices.

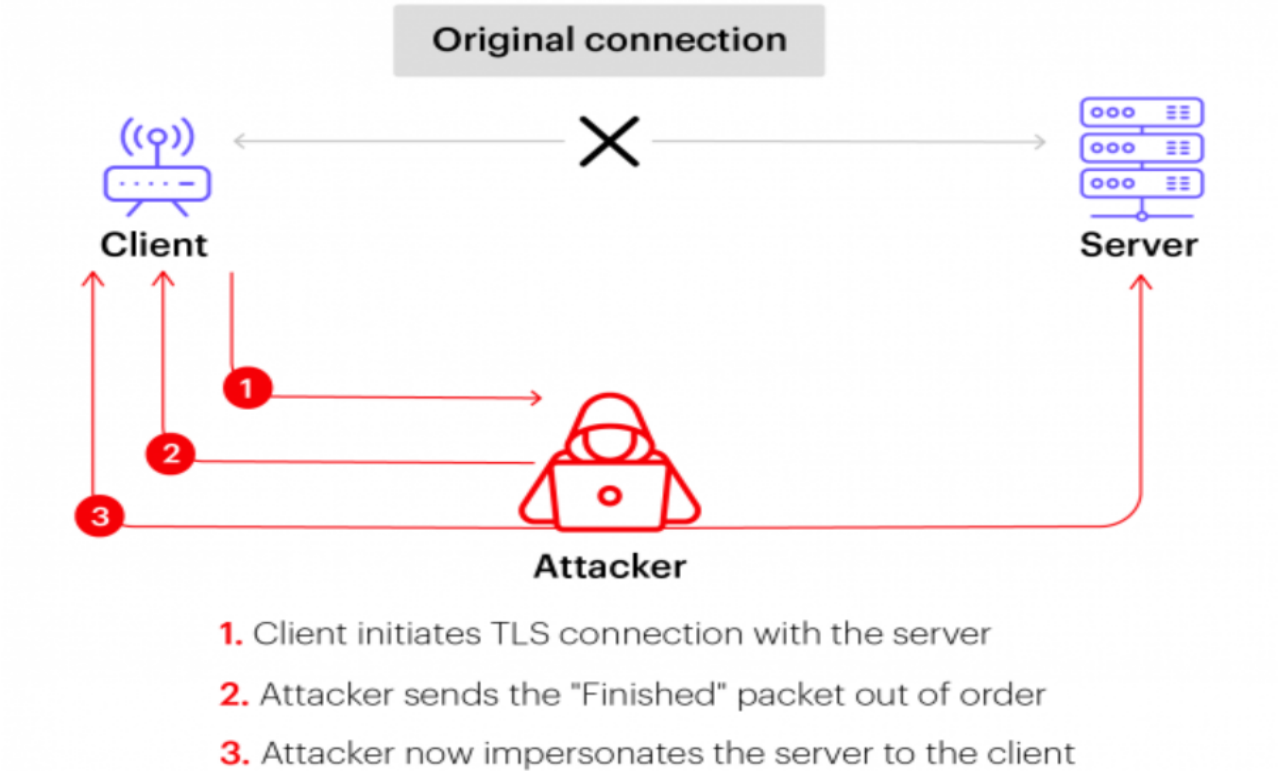
The vulnerability occurs in the set-up phase of a TLS session, called a "handshake", which involves the client and server exchanging several predefined messages in order to identify themselves and agree on cryptographic keys and functions to use during the session.

CVSS Score: 6.8 / Medium

Description

The vulnerability exists due to incorrect implementation of TLS 1.3 protocol within the SanityCheckTls13MsgReceived() in tls13.c when processing server data in the WAIT_CERT_CR state. A remote attacker can perform a Man-in-the-Middle (MitM) attack and read or modify information passed between clients using the wolfSSL library and TLS servers.

An attacker who's capable of intercepting the client's attempt to contact a server using TLS 1.3 can respond with invalid packets during the TLS handshake phase. This is not so far-fetched, as there are many techniques that allow an attacker to redirect a client's packets to a host they control, including DNS poisoning and BGP protocol manipulations. Once there, the attacker can bypass the server authentication step using the vulnerability disclosed by NCC Group, an information assurance firm headquartered in Manchester, United Kingdom which assesses, develops, and manages cyber threats across our increasingly connected society. They advise global technology, manufacturers, financial institutions, critical national infrastructure providers, retailers, and governments on the best way to keep businesses, software, and personal data safe - by sending a Finished message earlier than expected during the TLS handshake.



Exploitation

In RFC 8446, Appendix "A.1. Client" summarizes the legal state transitions for the TLS 1.3 client handshake. wolfSSL does not strictly enforce the TLS 1.3 client state machine.

Specifically and in the case of server certificate authentication, the wolfSSL TLS client state machine accepts a "Finished" message in the "WAIT_CERT_CR" state, just after having processed an "EncryptedExtensions" message. This is incorrect according to RFC 8446. wolfSSL should accept only "CertificateRequest" or "Certificate" messages as valid input to the state machine in the "WAIT_CERT_CR" state.

This permits attackers in a privileged network position to completely bypass server certificate validation and authentication, therefore allowing them to impersonate any TLS servers to which clients using the wolfSSL library are connecting.

The vulnerability exists due to incorrect implementation of TLS 1.3 protocol within the SanityCheckTls13MsgReceived() in *tls13.c* when processing server data in the WAIT_CERT_CR state. The code snippet below shows how incorrectly the function was configured and this is a result of not following secure coding practices

A remote attacker can perform a Man-in-the-Middle (MitM) attack and read or modify information passed between clients using the wolfSSL library and TLS servers.

```
case finished:
        #ifndef NO_WOLFSSL_CLIENT
            if (ssl->options.side == WOLFSSL_CLIENT_END) {
                if (ssl->options.clientState < CLIENT_HELLO_COMPLETE) {
                    WOLFSSL_MSG("Finished received out of order");
                    return OUT_OF_ORDER_E;
                }
                if (ssl->options.serverState <
                                    SERVER_ENCRYPTED_EXTENSIONS_COMPLETE) {
                    WOLFSSL_MSG("Finished received out of order");
```

```
            return OUT_OF_ORDER_E;

        }

    }

#endif
```

Impact

An attacker in a privileged position can read or modify communications between clients using the wolfSSL library and TLS 1.3 servers.

The attacker was able to impersonate the server for all possible purposes, including exposure of any information sent by the client, and taking control of the client if it receives any commands or software from the server. The attacker was also able to impersonate the client to the server, creating a separate connection while potentially using any legitimate messages sent by the client.

Mitigation

Users of the wolfSSL library for TLS 1.3 clients should update to the latest version of wolfSSL.

If a version upgrade is not possible, there are two mitigating steps you can take:

1. Apply the original patch and rebuild wolfSSL from the source.
2. Configure wolfSSL to remove support for TLSv1.3. This is done by removing the --enable-tls13 configuration option from the build settings (it is disabled by default) and rebuilding wolfSSL from the source.

There is another option for mitigating the vulnerability without modifying the device code: configuring the server to enforce TLS client authentication. This will force the client to authenticate using a certificate. As a result, the attacker will not be able to impersonate the client to the server (the attacker can still impersonate the server to the client).

Reference

1. https://research.nccgroup.com/2020/08/24/technical-advisory-wolfssl-tls-1-3-client-man-in-the-middle-attack/
2. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-24613
3. https://www.vdoo.com/blog/shaking-hands-wolves-wolfssl-cve-2020-24613-vulnerability

6.5.5 CVE-2020-1971: NULL pointer dereference bug in OpenSSL cause crash while verification of certificate.

Introduction

OpenSSL is a robust, commercial-grade, and full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library.

The core library, written in the C programming language, implements basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.

A null pointer dereference flaw was found in openssl. A remote attacker, able to control the arguments of the GENERAL_NAME_cmp function, could cause the application, compiled with openssl to crash resulting in a denial of service.

CVSS Score: 6.9 / Medium

Description

A NULL pointer dereference occurs when the application dereferences a pointer that it expects to be valid, but is NULL, typically causing a crash or exit.
NULL pointer dereference issues can occur through a number of flaws, including race conditions, and simple programming omissions.

NULL pointer dereferences usually result in the failure of the process unless exception handling (on some platforms) is available and implemented. Even when exception handling is being used, it can still be very difficult to return the software to a safe state of operation.

The v3_genn.c file of openssl's crypto module (crypto/x509v3/v3_genn.c) is responsible for handling X.509 certificate version 3 processing.

The GeneralName type is a generic type for representing different types of names. One of those name types is known as EDIPartyName. The function GENERAL_NAME_cmp which compares different instances of a GENERAL_NAME to see if they are equal or not. This function behaves incorrectly when both GENERAL_NAMEs contain an EDIPARTYNAME. A NULL pointer dereference and a crash may occur leading to a possible denial of service attack.

This was fixed by the openssl team by writing a wrapper function to safely compare edipartyname.

```
static int edipartyname_cmp(const EDIPARTYNAME *a, const EDIPARTYNAME *b)
```

The function ensures if the value of the two nameAssigner variables are NULL, then the function returns -1 to avoid null pointer dereferencing.

```
   if (a->nameAssigner == NULL && b->nameAssigner != NULL)
       return -1;
   if (a->nameAssigner != NULL && b->nameAssigner == NULL)
       return 1;
```

```
-snip-
/*
* partyName is required, so these should never be NULL. We treat it in
* the same way as the a == NULL || b == NULL case above
*/
 if (a->partyName == NULL || b->partyName == NULL)
     return -1;
```

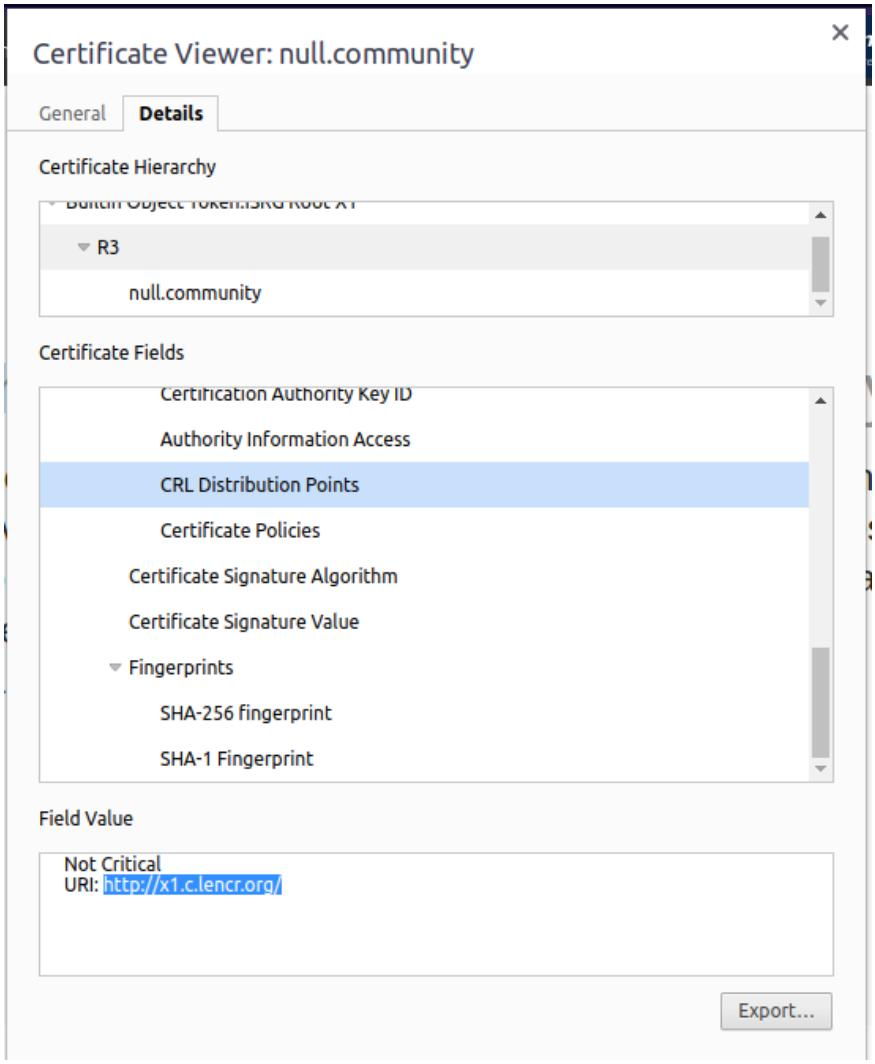
Source : openssl github

Some applications automatically download CRLs based on a URL embedded in a certificate. This checking happens prior to the signatures on the certificate and CRL being verified. OpenSSL's s_server, s_client and verify tools have support for the "-crl_download" option which implements automatic CRL downloading and this attack has been demonstrated to work against those Tools.

Exploitation

OpenSSL itself uses the GENERAL_NAME_cmp function for two purposes:

1. Comparing CRL distribution point names between an available CRL and a CRL distribution point embedded in an X509 certificate.



the Certificate Revocation List (CRL) point as seen ins the X509 of null.co.in

2. When verifying that a timestamp response token signer matches the timestamp authority name (exposed via the API functions TS_RESP_verify_response and TS_RESP_verify_token).

If an attacker can control both items being compared, a crash might be triggered.

For example, the attacker can trick a client or server into checking a malicious certificate against a malicious CRL.

Impact

Successful exploitation of this vulnerability could lead to Denial of Service (DoS).

Mitigation

The OpenSSL Project has released a version of OpenSSL to address this vulnerability. Please consider applying the update after thorough testing.

Applications not using the GENERAL_NAME_cmp of OpenSSL are not vulnerable to this flaw.

Reference

- https://www.openssl.org/news/secadv/20201208.txt
- https://nsfocusglobal.com/openssl-denial-of-service-vulnerability-cve-2020-1971-threat-alert/
- https://access.redhat.com/security/cve/cve-2020-1971

6.5.6 CVE-2020-0688: Remote Code Execution on Microsoft Exchange Server through fixed cryptographic keys

Introduction

A remote code execution vulnerability exists in Microsoft Exchange Server when the server fails to properly create unique keys at install time.

Knowledge of the validation key allows an authenticated user with a mailbox to pass arbitrary objects to be deserialized by the web application, which runs as SYSTEM.

CVSS Score: 8.8 / High

Description

Instead of generating random keys at the time of installation, all installations of Microsoft Exchange server have the same validation Key and decryption Key values in web.config. These keys are used to provide security for ViewState. Due to the static nature of the keys, authenticated users can easily trick the exchange server into deserializing maliciously crafted payload data resulting in complete compromise of the target server.

Exploitation

The vulnerability is found in the Exchange Control Panel (ECP) component.

View State is server-side data that ASP.NET web applications store in serialized format on the client side. The client provides this data back to the server via _VIEWSTATE request parameter.

```
  <system.web>
    <machineKey validationKey="CB2721ABDAF8E9DC516D621D8B8BF13A2C9E8689A25303BF"
decryptionKey="E9D2490BD0075B51D1BA5288514514AF" validation="SHA1"
decryption="3DES" />
    <!--
```

Fig. Exchange Vulnerability: Web.Config containing static validation key

Due to the use of static keys , an authenticated attacker can trick the server into deserializing maliciously crafted View State data.

YSoSerial.net is a tool that can execute arbitrary .NET code on the server in the context of the Exchange Control Panel web application, which runs as SYSTEM.

Steps:

1. Collect ViewStateUserKey and _VIEWSTATEGENERATOR values from an authenticated session.
    a. ViewStateUserKey is the Session ID of the user. It can be obtained from the cookie value using F12->Cookie value->Session ID.
    b. Using document.getElementById("_VIEWSTATEGENERATOR"). Value any one can obtain _VIEWSTATEGENERATOR value.
2. With the help of all 4 parameters,
    a. validationKey
    b. validation algorithm
    c. _VIEWSTATEGENERATOR
    d. ViewStateUserKey

    The next step is to create a payload:

    ysoserial.exe -p ViewState -g TextFormattingRunProperties -c "echo > c:/Vuln_Server.txt" --validationalg="SHA1" --validationkey="CB2721ABDAF8E9DC516D621D8B8BF13A2C9E8689A25303BF" --generator="B97B4E27" --viewstateuserkey="05ae4b41-51e1-4c3a-9241-6b87b169d663" --isdebug –islegacy

Submitting the above payload on the internet results in creating a vuln_Server.txt file under C:\ with SYSTEM privileges. This means any authenticated user can execute malicious files on the server.

Impact

Any authenticated user can execute the malicious payload on the Exchange server resulting in the total server compromise.

Mitigation

Validation key is the security provided to the **_VIEWSTATEGENERATOR** and _SessionID that is Validation User Key parameter.

Instead of being static, It needs to be randomly generated. This will result in protecting the serialization. The key needs to be random at all times.

Reference

1. CVE-2020-0688
2. https://www.zerodayinitiative.com/blog/2020/2/24/cve-2020-0688-remote-code-execution-on-microsoft-exchange-server-through-fixed-cryptographic-keys

6.5.7 CVE-2019-14855: Chosen-prefix collision for SHA-1

Introduction

A Chosen-prefix collision attack is an extension of the collision attack. In this case, the attacker can choose two arbitrarily different documents, and then append different calculated values that result in the 2 documents having an equal hash value. This attack is much more powerful than a classical collision attack.

Mathematically stated, given two different prefixes *p1, p2*, the attack finds two appendages *m1* and *m2* such that *hash(p1 ∥ m1) = hash(p2 ∥ m2)* (where ∥ is the concatenation operation).

CVSS Score: 7.3 / High

Description

Researchers Gaëtan Leurent and Thomas Peyrin in January 2020 presented : "SHA-1 is a Shambles", The first practical chosen-prefix collision attack on SHA-1 hashing algorithm. They used 900 Nvidia GTX 1060 GPUs and two months of computation to achieve this collision.

SHA-1 is a cryptographic hashing scheme based on Merkle-Damgard construction, which gives a 160 bit hash value. In particular MD contractions are known for chosen prefix collision and length extension attacks as they compute the hashing rounds on a fixed size output buffer over multiple cycles. It is also understood that achieving these collision in practice is difficult and computationally intensive. After the announcement of "SHA-1 is a Shambles", its totally unsafe to use SHA-1.

Setup Phase: Find a set of "nice" chaining value differences S, where S is a set of 2^38.
This phase introduces the complexity about sqrt(2^n/|S|) = 2 61. The chaining of iterations is done to reduce the memory. This truncates SHA-1 to 96 bits and makes it easy to find a partial collision in S.

Birthday phase: Find m1,m0 such that H(P1 k m1) − H(P2 k m0) ∈ S

First step, each GPU computes independently a series of chains, and in the second step we gather all the results, sort them to find collisions in the end-points, and re-run the chain to locate the collisions. Every time we run the second step, we then search the collisions in the graph, to determine whether we have reached a useful starting point.

Near-collision phase: Erase the state difference, using near-collision blocks
The near-collision phase is very technical and very complex. Every time a block is found, we have to prepare the search for the next block. This first requires traversing the graph G to find the parameters for the next block: we have different constraints in the last steps depending on which output differences are desired. Then, we had to generate a new non-linear part for the early steps. Finally, some testing and configuration of the GPU code was then required to check how neutral bits and boomerangs behaved in this new configuration. In particular, there were usually some adjustments to make in the. GPU code for the more complex conditions in the path that involve several bit positions.

Complexity for breaking the hash 2^63 ~ 2^64.

Exploitation

Impersonation attacks can be launched when this flaw is exploited.

- Bob creates keys such that. H(Alice||kA) = H(Bob||kB)
- Bob asks CA to certify his key kB
- Bob copies the signature to kA, impersonates Alice

Steps:

1. Build a chosen-prefix collision with prefixes “99 04 0d 04 ** ** ** ** 01 20 00” and “99 03 0d 04 ** ** ** ** 01 18 00”, after filling the ** with two arbitrary time-stamps. The chosen-prefix collision must have at most 10 near-collision blocks. This determines the ?? bytes of the keys.
2. Choose a tiny JPEG image to include in key B (fixed orange bytes), and an arbitrary UserID to include in key A (fixed yellow bytes).
3. Select “!!” bytes in B to obtain a modulus with known factors
4. Select “!!” bytes in A to obtain a modulus with known factors
5. Generate key B with the modulus and the padded JPEG. Ask for a signature of the key.
6. Copy the signature to key A.

Impact

- SHA-1 certificates (X.509) still exists, the CAs sell legacy SHA-1 certificates for legacy clients,
- PGP signatures with SHA-1.
- GnuPGv1
- 1% of public certifications
- SHA-1 still allowed for in-protocol signatures in TLS, SSH
- 3% of Alexa top 1M servers
- Many Github instances still track file version using SHA1 fingerprints

Mitigation

1. SHA-1 must be deprecated
2. Choose the hashing algorithm for which no theoretical or practical attacks till now.

Reference

- https://sha-mbles.github.io/
- https://eprint.iacr.org/2020/014.pdf
- https://nvd.nist.gov/vuln/detail/CVE-2019-14855

6.5.8 CVE-2020-2655 JSSE Client Authentication Bypass

Introduction

In most deployments of TLS, only the server authenticates itself. It usually does this by sending an X.509 certificate to the client and then proving that it is in fact in possession of the private key for the certificate or computing the shared secret or by signing the ephemeral public key of the server.

The client usually does not authenticate itself within the TLS handshake, but rather authenticates in the application layer. However, TLS also offers the possibility for client authentication during the TLS handshake. It follows the same mechanism like it is present in DTLS (Datagram Transport Layer Security protocol).

If the Client does not possess a certificate the client is then supposed to send an empty certificate and skip the CertificateVerify message. It is then up to the TLS server to decide what to do with the client. Some TLS servers provide different options in regards to client authentication and differentiate between *REQUIRED* and *WANTED* and *NONE*. If the server is set to *REQUIRED*, it will not finish the TLS handshake without client authentication. In the case of *WANTED*, the handshake is completed and the authentication status is then passed to the application.

If somehow we can get authenticated even though the server is set to REQUIRED without providing proof that we own the private key for the Certificate we transmitted then this kind of issue is known as Authentication bypass.

CVSS Score: 4.8 / Medium

Description

The Java Secure Socket Extension, which is the SSL/TLS part of JCE uses a producer/consumer architecture to decide on which messages to process.

- JSSE DTLS/TLS Server accepts the following message sequence, with client authentication set to **REQUIRED**.

```
ClientHello -------->

                                ServerHello

                                Certificate

                                ServerKeyExchange*

                                CertificateRequest
```

<-------- ServerHelloDone

ClientKeyExchange

[ChangeCipherSpec]

Finished

-------->

[ChangeCipherSpec]

<-------- Finished

JSSE is totally fine with the messages and finishes the handshake although the client does NOT provide a certificate nor a CertificateVerify message. It is also ready to exchange application data with the client.

- Authentication of a user to a DTLS server with the following message sequence:

ClientHello -------->

ServerHello

Certificate

ServerKeyExchange*

CertificateRequest

<-------- ServerHelloDone

ClientKeyExchange

[ChangeCipherSpec]

Certificate

Finished

[ChangeCipherSpec]

Finished

```
        -------->

                    [ChangeCipherSpec]

                <-------- Finished
```

Instead of the Finished message, we send a Certificate message, followed by a Finished message, followed by a second(!) CCS message, followed by another Finished message. Somehow this sequence confuses JSSE such that we are authenticated without providing the proof that we own the private key for the Certificate we transmitted.

Exploitation

To exploit the described vulnerabilities, you have to send (D)TLS messages in an unconventional order or have to not send specific messages but still compute correct cryptographic operations. To do this, you could either modify a TLS library of your choice to do the job.

Using TLS-Attacker (https://github.com/tls-attacker/TLS-Attacker) and with a custom XML workflow file with the specific messages we want to transmit this attack can be performed. TLS-Attacker was built to send arbitrary TLS messages with arbitrary content in an arbitrary order - exactly what we need for this kind of attack.

Impact

This bug is found in Java 11 and Java 13 (Oracle and OpenJDK) and does bug does not exist in the older versions.

These bugs are quite fatal for client authentication. The vulnerability got CVSS:4.8 as it is "hard to exploit" apparently.

Mitigation

1. Use the latest java version of Oracle and OpenJDK
2. Remove the CertificateVerify handshake consumer so the state machine doesn't expect it.
3. For empty Certificate messages, we should not expect a CertificateVerify message to follow
4. Ensure that the CertificateVerify message follows the Certificate key exchange.
5. Make sure that any expected CertificateVerify message has been received and processed.

Reference

- https://web-in-security.blogspot.com/2020/01/cve-2020-2655-jsse-client.html

- https://github.com/tls-attacker/TLS-Attacker

6.5.9 Netgear Signed TLS Cert Private Key Disclosure

Introduction

The private key of any certificate should remain private. However, if not maintained the secrecy of the private key could lead to complete compromise of the device, as the complete security of the device depends on securely keeping the private key of the certificate. This vulnerability explains the weakness of Netgear , not maintaining the TLS certificate private key and exploitability and discoverability factors of finding and compromising the device by using private key as it is available to the public are highest and is a critical vulnerability finding for NetGear.

CVSS Score: 7.5 / High

Description

There are at least 2 valid signed TLS certificates that are bundled with publicly available Net gear device firmware. These certificates are trusted by browsers on all platforms but will surely be added to revocation lists shortly. The firmware images that contained these certificates along with their private keys were publicly available for download without authentication, Thus any one in the world could have retrieved these keys.

Exploitation

As Net gear signed TLS certificates are publicly available with private keys, anyone could download the same with private keys.

There are 2 certificates with private keys.

Both keys found were contained in the **R9000-V1.0.5.8** firmware image file available here:

http://www.downloads.netgear.com/files/GDC/R9000/R9000-V1.0.5.8.zip

The certificate which is signed by **EnTrust** is valid for the following DNS host names:

- www.routerlogin.net
- www.routerlogin.com
- Routerlogin.com
- Routerlogin.net

The corresponding private key for the certificate is:

-----BEGIN PRIVATE KEY-----

MIIEvwIBADANBgkqhkiG9w0BAQEFAASCBKkwggSlAgEAAoIBAQDJbQp5ycpkzerw
rPS/9jcrG1r5LBBS1u5OIdMAKxj9CKBrkiaGQCbv87fcTWO4BA6geLyHTlDWLTzU
8e24EJy/5+tZqRlN99xznbATCilBTkclJREYZINnvXdtIrMf39spCSDZoT9nlYPO
fgLIb0ZG9WAbdTCK3MSg5uOX4PXXzhUhLSbC72ameb+sKK/i1H9ujTGlB/rG4pHM
sM/lJ0zxjdgUi8rVxysQchJmY0YCG/KriqIcGDkfTO2pyu3nBZamaqCrdr1oxu5D
TeRRznmjC4F86meHdQMl7l/5Z9cSqHbFozc1X9FhJqua87N9TdEkc+3XdD7oudVO
15+18kbFAgMBAAECggEAXO3t01qCAhfuuLNTB+10bnLkeekWbuGyeGGqk31Ovg7o
1DhNhcq7LCFpRj5+LzvP88FAWbyMFwQv+J7VagJgDznUnz0g65PbJYwu29noRrTy
pfR/+p0E8yu6maNUuPPyjPAlM8LtGcElWnNQnWCZL8utV32ts6M/JGzhvASR58ne
5M6fB3RPgfoe50zaBu7lQ5YhlNpj8J0P1eg0gl/535nx2FqejqqkBu10xfo3T3dA
KdJluNA3rU9BHjzYl/+8SKZ9wYekuf79CM+zSrVe2iJ7gZP2V3XhUdtKgL8ZgNHG
TNW4LOlOvG7/4XH+GkacspjC5ZA/OOlx/yqXzkE/UQKBgQDoyRhy5jBF0TNPrR1A
smQK+t1eEl27LbM7cFnldlbGa/gP+3U8zPMBGru7fdED9dVzJ+2p6lzHkDAP3KcT
ftOleABWtGWZ724whxDu996BeXl0lnWjY8dEUT3aod9lpC/PUiCV+MmUSTPBFXMl
bF5ZA+dpbmp7IBP/7FIdRveuPwKBgQDdg1mW2sm0GdeEETu3oeK4Xi+7X63jhc33
VsBJ5285RmQWLPbMAANirHIQ+mTU4XIn24LNfugteT5ocJlY7TjTje4ldNY5k0Ha
9tJYMD3ZnmT++NEFM919vaelCoGMj0K2KUYZ4ef/IT9iisUA+Wz9HL5So99J/RFw
Hhr5D+fR+wKBgQCVAGsq2Jabid27KUbpK4aH1K2vUQ83eXgZGsAf9VBz75Y3vK/9
O/5rfY4e49jPHSMEc9FXipDamDt7W7SB8RM7bfxhg1TpZG12mG3JWFVPMMpeSNSB

whNBcnMSJ7zT1XVY0evTswxsYzLCa5VppeT6O1p7jNaReyXyEXU6EjvlhwKBgQCW
rBuqUxPUH6fKu3YlSEZRJi/Ngh1jn8YjsayGGJg9GzZFJzyQMooa+jV0ev1PGDJw
Dg4A/YusMsZSgBSuuI1m+Sm97KUy8IlhCXa2acoIVodYL6LtqQPF3dVEm8rW8zNr
eNOoE1Oc5N6ahs3OBdsup/iFedYOG8davf+W3kzPNQKBgQCVJ1ehlnFwT8bFs+PU
hgrLHrvh6XHqcdJ1NOR52Su5Ge0ZdaAFtisSoJm2bVpbfyFJbfjMRg1FdhZr71Kb

LLG0IuutcnTua/FicLhUjFT5qrfs9he8trAhMgjT4t4HCH0H19JoaEOUUU8FMPLi

QZbodRspSr2dBf316Kh1N8C8Yg==

-----END PRIVATE KEY-----

These files were found respectively at /etc/uhttpd.crt and /etc/uphttpd.key in the aforementioned firmware image.

In the same firmware image was an additional valid TLS Certificate and its corresponding private key.

The files were found at the paths **/data/funjsq/config/httpd/cert.pem** and **/data/funjsq/config/httpd/key.pem**.

The corresponding private key is:

-----BEGIN PRIVATE KEY-----

MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDMV9VFrb1gyK9v

UJfv32ewHWkdo3XpugiNTFQutoMcLuEfZgz7etCazlKkP3Vw9Dlz8/WGLpZZ5qVU

YhdSFa1vIr19hDYUejscYGF6fYbY4plJ1AbpegDyQ/kRhwbIIA78FVG7E53tJznf

zexGb+2nVk9xo0bXJfJaOKkjoYkObvE9awQFDosyvPEcDvhsleLNbDga5qY/OyJB

9yNFNoJYPKWJqm4W4DLCOKhCut6utAPzC6Sea6YxaBTaIJOqp6dJ9G0+yDly4WI1

zGc/CC6ui6z8FDuduMdbm9sIOiuYqg4+klzn5tsTu0fgOz1g5fMi6I8BBM/jw/x+

4mwjLEgXAgMBAAECggEANCH0d2Jr/IU07OFS4g/NFFFsj/M0Ef00UVg34fMOYBJV

vPz8MuySa+xXiS/ndnnYboy/Bwy7rxP4+h5MdNSy+reSQIOKiI7mpcaxF98OmCa3

lO5TFR3bP/O3h5E7WbNUH1wRDfljQS3QxhhzP0UvDJIokoVlfV5hBkOOY1jC7rtK

KtdInqD0y+ALkxDAj5yk/8kSxSirO5/3nzZAgzQxGr2r2psytEQlVMqRD4rcLymH

WQ2GeZe0gr9wcWiFDvFBc3S7Blh476BdNKbNuvplYdHq1PKnD1b2NegBIzYH+fb6

DE+XLCrSADwDbIE//L4B+fvK6eOciMPqvmUD2mPGAQKBgQD2Qcu7LexNlKbk3B8v

BBYwSda+GCclvnAAZdBrEtp++nuCHhusk6UokM4e3Xabq8//pFm9UDaGdmW0S+Bs

37TvcbWNidpJ7hEOS93sQ9dp/08GQKRb/tAfHRJ/GRgnd9tJKvuQwx/5PRBl4gJb

EROFG+zNnNSFXZU8i0pwxUig0QKBgQDUbYOMm7Py8sYSU5dLfCPCvLFdYCddppfh

7wwFduF1ipqTfhFejOy+2TnxHFOEBX96GfNRwtgP5FmKMEmbSYELEH7V+6yJus+k

MGxck6+pD2aX7bIUA96HOHCCOUA4zE6V/aN7nIon+jgtj2vkjB3m7FCv6Tu05jTS

C1wsXv9UZwKBgQDs2SGTGTsy7uuKKPDRLpQMw6gH04ErezuMFmDb6xk9kbrizgR9

+s+Z8ZRd+VFPrnNyhGdPfuvCbfOp8mSbMpp6xhoBVPofqxq6blu3FxUDvOwLrnam

iLExi6uzlLY3l77QY8frVtDUzleMNlFft0/X8aDTrah10SltPudNCWKuUQKBgEy2

e8IkXHjI6XSm0UVGQFfL8rVIyw6L4d+KhynWA5eCBU5sQXOGqtxE8CK9Wv+bSKzc

gD0vgv1CNn+r7njws3Q+cb9u0qEuYFvnzMol31LLJ/6HrT6DIzJr1F2CtUmNpneO

ECLMpivHtc/mMk1nuEizYHQWYyRx2fNcfN3dNJHTAoGBALeEHU0qn883Us/iyftC

mB4CqgWDJ22jhDWbpu/mdagCymRWuP8hPHXlGcMSBSo2fZz9TBx9p5dAJkXSbUd+

BiG73boMCjmNrJnB+06rh7lwcPIGpD3XAhQ14rm7YUC8Y7VAUGw6e/H51zOJ+eQx

xqGaW9IOaJ7fMqT2LMo1yENb

-----END PRIVATE KEY-----

The DNS host name for the certificate is **mini-app.funjsq.com**

Impact

As the private key of the certificate is available to the public. Anyone could do spoofing attacks using the certificates, MITM attack, session hijacking etc. to grab sensitive information, tamper the integrity of any sensitive data in the communication channel. This can be a very serious attack vector for the vulnerability.

Mitigation

Mentioned certificates need to be revoked so that major browsers do not trust them any longer.

Reference

- https://gist.github.com/nstarke/a611a19aab433555e91c656fe1f030a9

6.5.10 CVE-2020-8172: Node.js - Insecure TLS session reuse can lead to hostname verification bypass

Introduction

Node.js is a software development platform for building fast and scalable network applications in the JavaScript programming language.

The *tls* module uses OpenSSL to provide Transport Layer Security and/or Secure Socket Layer: encrypted stream communication.

TLS/SSL is a public/private key infrastructure. The server needs to have a private key in order to provide a legitimate identity to all the clients.

Multiple NetApp products incorporate Nodejs.
Nodejs versions 12.0.0 prior to 12.18.0 and 14.0.0 prior to 14.4.0 are susceptible to the vulnerability.

CVSS Score: 7.4 / High

Description

The Node.js TLS library supports client-side reuse of TLS sessions when multiple connections to the same server are opened.

Code that wants to use this feature can listen for the 'session' event on a *tls*.TLSSocket to get notified of newly created TLS sessions. The documentation for this event explicitly mentions that the passed sessions "can be used immediately or later".

The problem with this design is that 'session' events are triggered even if verification of the server certificate hostname in onConnectSecure fails.

onConnectSecure is triggered by the OpenSSL info callback (with the flag SSL_CB_HANDSHAKE_DONE) after a TLS handshake. The 'session' event is triggered by OpenSSLs get_session_cb, which can happen before the info callback in TLS 1.2 and after in TLS 1.3 and which is triggered regardless of the result of onConnectSecure.

This means that sessions where the server presented an invalid certificate, or one with a wrong hostname, will trigger the session event and can end up being reused or stored in a
cache.
This behaviour is insecure, because resumed sessions will not be subjected to another hostname verification check as long as they are CA signed:

```
// Verify that server's identity matches it's certificate's names
// Unless server has resumed our existing session
if (!verifyError && !this.isSessionReused()) {
const hostname = options.servername ||
options.host ||
(options.socket && options.socket._host) ||
'localhost';
const cert = this.getPeerCertificate(true);
verifyError = options.checkServerIdentity(hostname, cert);
}
```

In practice, this means that the immediate reuse described in the API documentation is always insecure and that session caches are at risk of storing insecure sessions. The most important implementation of a session cache is in the *https* library. New sessions are stored in the cache when the 'session' event is triggered and are evicted once a *tls* socket is closed with an error.

```
if (options._agentKey) {
// Cache new session for reuse
socket.on('session', (session) => {
this._cacheSession(options._agentKey, session);
});
// Evict session on error
socket.once('close', (err) => {
if (err)
this._evictSession(options._agentKey);
});
}
```

This opens a small race window where an invalid session can be used by other *HTTPs* requests to the same host. The proof-of-concept wins the race reliably against a local server using a setImmediate() callback, but there are probably other ways this could be exploited in real-world applications.

Exploitation

The exploit requires a target server with a valid CA-signed certificate (for an arbitrary hostname) and support for TLS resumption. A minimal *golang https* server that worked was used here.

Below is the terminal output when you run the poc.js code which is the exploit written in Javascript. It starts sending requests to the host and once it finds a suitable request, then a small race window is opened where an invalid session can be used by other HTTPs requests to the same host, and thus this can lead to hostname verification bypass.

```
[fwilhelm@fwilhelm node]$ ../node/node-v13.9.0-linux-x64/bin/node poc.js

[!] First request failed:Host: nodejs.org. is not in the cert's altnames: DNS:loca.host

[x] Starting second request

[x] Dumping globalAgent._sessionCache.map:

{

'nodejs.org:8444::::::::::::::::TLSv1_2_method:': <Buffer 30 82 06 2f 02 01 01 02 02 03
04 04 02 13 01 04 20 cd b7 17 84 ac 9f 31 6f 1c cc 73 de 31 05 eb dc 60 62 df c7 c5
d5 8c b4 75 cc a7 28 1f d9 c0 22 04 ... 1537 more bytes>

}

[!] Bypassed hostname verification. Server response: 200

{

date: 'Thu, 05 Mar 2020 17:08:24 GMT',

'content-length': '29',

'content-type': 'text/plain; charset=utf-8',

connection: 'close'
```

```
}
```

Impact

Successful exploitation of this vulnerability could lead to the disclosure of sensitive information or the addition or modification of data.

Mitigation

Users should upgrade Node.js to the latest version available.

Reference

- https://nodejs.org/en/blog/vulnerability/june-2020-security-releases/
- https://bugs.chromium.org/p/project-zero/issues/detail?id=2019

6.5.11 CVE-2020-13777: GNU TLS 1.3 session resumption works without a master key, allowing MITM

Introduction

GnuTLS is a secure communications library implementing the SSL, TLS, and DTLS protocols and technologies around them. It provides a simple C language application programming interface (API) to access the secure communications protocols as well as APIs to parse and write X.509, PKCS #12, OpenPGP, and other required structures.

A flaw was reported in the TLS session ticket key construction in GnuTLS, a library implementing the TLS and SSL protocols. The flaw caused the TLS server to not securely construct a session ticket encryption key considering the application-supplied secret, allowing a man-in-the-middle attacker to bypass authentication in TLS 1.3 and recover previous conversations in TLS 1.2.

CVSS Score: 7.4 High

Description

GnuTLS 3.6.x before 3.6.14 uses incorrect cryptography for encrypting a session ticket (a loss of confidentiality in TLS 1.2, and an authentication bypass in TLS 1.3). The earliest affected version is 3.6.4 (2018-09-24) because of an error in a 2018-09-18 commit. Until the first key rotation, the TLS server always uses the wrong data in place of an encryption key derived from an application.

GnuTLS servers were able to use tickets issued by each other without access to the secret key is generated by gnutls_session_ticket_key_generate(). This allows a MITM server without valid credentials to resume sessions with a client that first established an initial connection with a server with valid credentials.
Because the ticket can be used for resumption without knowledge of the master key it was assumed (but haven't been tested yet) that it can also be used for passive decryption of early data.

This issue was first noticed with Ubuntu version 3.6.13-2ubuntu1 and reproduced with a build from master as of 52e78f1e.

Exploitation

Steps:

1. Start a server with valid credentials: gnutls-serv --x509keyfile=authority/server/secret.key --x509certfile=authority/server/x509.pem
2. Connect to the server and store resumption data: openssl s_client -connect localhost:5556 -CAfile authority/x509.pem -verify_return_error -sess_out session.cache
3. Stop the server started in step 1.
4. Start a server with bogus credentials at the same address port (imagine a real attacker redirecting connections only if the client is attepting resumption): gnutls-serv --x509keyfile=rogueca/mitm/secret.key --x509certfile=rogueca/mitm/x509.pem
1. Connect again, using the stored resumption data: openssl s_client -connect localhost:5556 -CAfile authority/x509.pem -verify_return_error -sess_in session.cache

A method of using openssl s_client can also be used to reproduce the problem because gnutls-cli lacks a way to store resumption data across invocations, but the effect is also reproducible with applications using GnuTLS that cache session data long enough to change the server. The issue was noticed while implementing session resumption for proxy connections in mod_gnutls.

Impact

A remote attacker could recover previous conversations in TLS 1.2 and obtain sensitive information or conduct a man-in-the-middle attack to bypass authentication in TLS 1.3.

Successful exploitation of this vulnerability could lead to the disclosure of sensitive information or the addition or modification of data.

Mitigation

1. The problem can be corrected by updating your system to the following package versions:

Ubuntu 20.04

- libgnutls30 - 3.6.13-2ubuntu1.1

6.5.11.1.1 Ubuntu 19.10

- libgnutls30 - 3.6.9-5ubuntu1.2

1. All GnuTLS user should upgrade to the latest version:

```
# emerge --sync
# emerge --ask --oneshot --verbose ">=net-libs/gnutls-3.6.14"
```

Reference

- https://security.gentoo.org/glsa/202006-01
- https://gitlab.com/gnutls/gnutls/-/issues/1011
- https://nvd.nist.gov/vuln/detail/CVE-2020-13777
- https://www.debian.org/security/2020/dsa-4697

6.5.12 Go Crypto libraries panic during the recursive division of very large numbers CVE-2020-28362

Introduction

A vulnerability was found in Google Go up to 1.14.11/1.15.3 (Programming Language Software). It has been classified as critical. This affects some unknown functionality. The manipulation with an unknown input leads to a weak authentication vulnerability. CWE is classifying the issue as CWE-295. This is going to have an impact on availability.

The weakness was released on 11/19/2020. It is possible to read the advisory at lists.apache.org. This vulnerability is uniquely identified as CVE-2020-28362 since 11/09/2020. The exploitability is told to be easy. It is possible to initiate the attack remotely. No form of authentication is needed for exploitation. The technical details are unknown and an exploit is not publicly available. The pricing for an exploit might be around USD $0-$5k at the moment. The attack technique deployed by this issue is T1587.003 according to MITRE ATT&CK.

CVSS Score: 7.5 / High

Description

A number of math/big.Int methods (Div, Exp, DivMod, Quo, Rem, QuoRem, Mod, ModInverse, ModSqrt, Jacobi, and GCD) can panic when provided crafted large inputs. For the panic to happen, the divisor or modulo argument must be larger than 3168 bits

(on 32-bit architectures) or 6336 bits (on 64-bit architectures). Multiple math/big.Rat methods are similarly affected.

crypto/rsa.VerifyPSS, crypto/rsa.VerifyPKCS1v15, and crypto/dsa.Verify may panic when provided crafted public keys and signatures. crypto/ecdsa and crypto/elliptic operations may only be affected if custom CurveParams with unusually large field sizes (several times larger than the largest supported curve, P-521) are in use. Using crypto/x509.Verify on a crafted X.509 certificate chain can lead to a panic, even if the certificates don't chain to a trusted root. The chain can be delivered via a crypto/tls connection to a client, or to a server that accepts and verifies client certificates. net/http clients can be made to crash by an HTTPS server, while net/http servers that accept client certificates will recover the panic and are unaffected.

Moreover, an application might crash invoking crypto/x509.(*CertificateRequest).CheckSignature on an X.509 certificate request or during a golang.org/x/crypto/otr conversation. Parsing a golang.org/x/crypto/openpgp Entity or verifying a signature may crash. Finally, a golang.org/x/crypto/ssh client can panic due to a malformed host key, while a server could panic if either PublicKeyCallback accepts a malformed public key, or if IsUserAuthority accepts a certificate with a malformed public key.

Exploitation

Adversaries may create self-signed SSL/TLS certificates that can be used during targeting. SSL/TLS certificates are designed to instill trust. They include information about the key, information about its owner's identity, and the digital signature of an entity that has verified the certificate's contents are correct. If the signature is valid, and the person examining the certificate trusts the signer, then they know they can use that key to communicate with its owner. In the case of self-signing, digital certificates will lack the element of trust associated with the signature of a third-party certificate authority (CA).

Adversaries may create self-signed SSL/TLS certificates that can be used to further their operations or even enabling Adversary-in-the-Middle if added to the root of trust. After creating a digital certificate, an adversary may then install that certificate on infrastructure under their control.

Impact

Versions of geth built with Go <1.15.5 or <1.14.12 are most likely affected by a critical DoS-related security vulnerability.

The DoS issue can be used to crash all Geth nodes during block processing, the effects of which would be that a major part of the Ethereum network went offline.

Mitigation

Rebuilding with go 1.15.5 or 1.14.12 will suffice to address the vulnerability.

Reference

- https://blog.ethereum.org/2020/11/12/geth_security_release/
- https://github.com/advisories/GHSA-m6gx-rhvj-fh52
- https://vuldb.com/?id.165111
- https://github.com/golang/go/issues/42552

6.5.13 SSRF attack using TLS features, research presented at DEFCON safe mode

Introduction

A researcher Joshua Maddux of security firm Latacora, at DEFCON and BlackHat 2020 shows the session resumption provision in TLS protocol can be abused, in combination with DNS rebinding, to send crafted response by a malicious server to carry out SSRF attack, leading victim to send spams or phishing emails, or carrying out DDoS attacks.

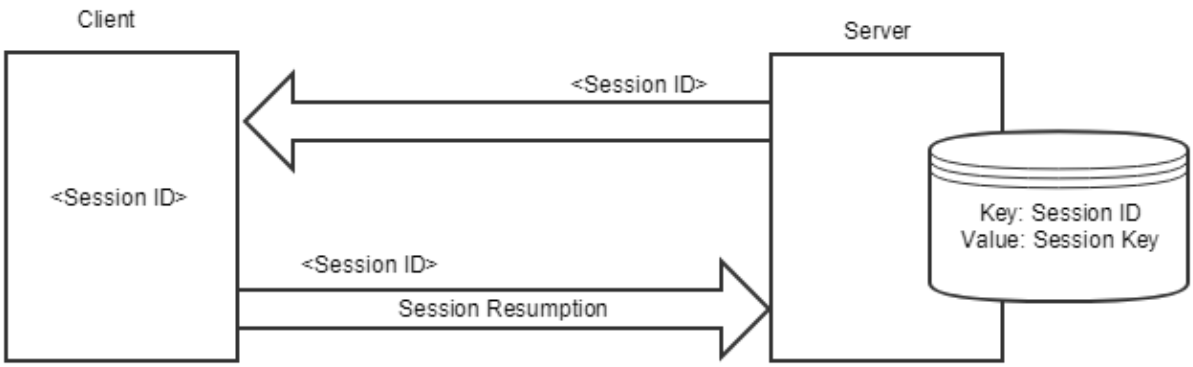
CVSS Score: 6 / Medium

Description

Joshua Maddux in his presentation used 3 major concepts to build his SSRF attacks. TLS resumption, DNS Rebinding & SSRF.

TLS Session Resumption

TLS protocol allows clients and servers to negotiate a token-value, that can be a **Session ID** (32 bytes in size) or **Session Ticket** as a part of TLS handshake.

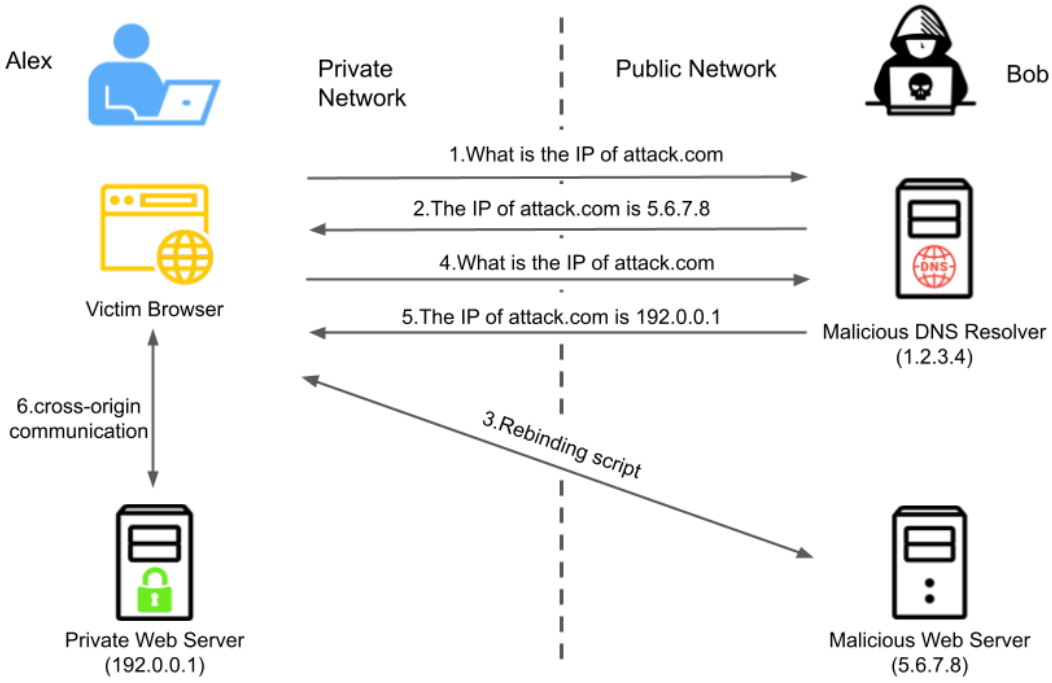


Source Compass Security

This value is used to avoid costly TLS handshake if the clients and servers want to resume the previously established session identified with the same token value.

DNS Rebinding

Independent to TLS, DNS rebinding is an attack method in which a malicious web page causes visitors to run a client-side script, which makes a new DNS request for the domain, to which it is already connected, which is controlled by an attacker.



Source - [Palo Alto]

The attacker replies with a new IP address. For instance, they could reply with an internal IP address or the IP address of a target somewhere else on the Internet

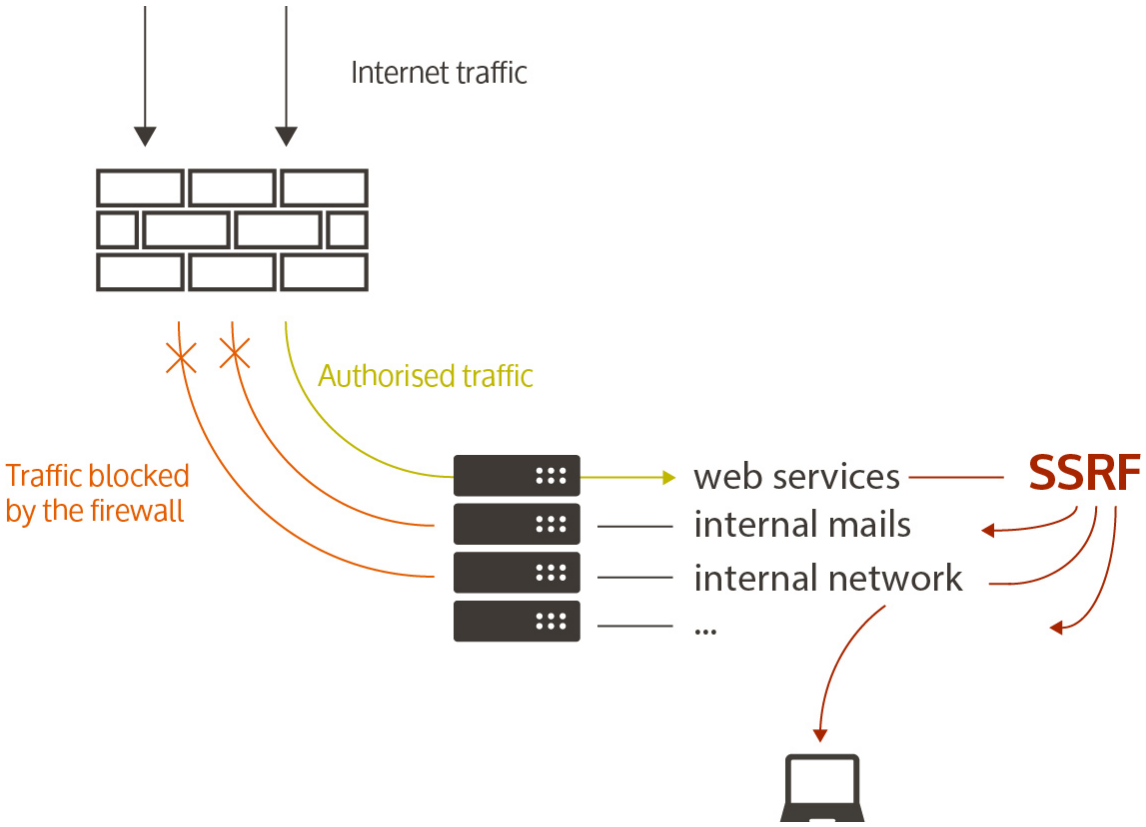
This is a commonly used attack to breach a private network by causing the victim's web browser to access computers at private IP addresses and return the results to the attacker.

Server Side Request Forgery (SSRF)

SSRF is a web security vulnerability that allows an attacker to trick the server-side application to make HTTP requests to an arbitrary domain of the attacker's choosing.

In a typical SSRF attack, the attacker might cause the server to make a connection to

internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, potentially leaking sensitive data such as authorization credentials.



Source Vaadata

Joshua used the TLS connection requests to connect to a vulnerable server. Once a connection is established, the session id obtained is used to make a deferred-persistent session to another local service like SMTP, or memcache, using DNS binding attack. Over multiple connection resumption requests, SSRF payloads are sent to the targeted applications. These applications are tricked to work on behalf of the attacker.

Exploitation

Aim of Joshua's presentation was to trick a user to send a phishing email from a local SMTP server which is not accessible to an attacker from outside of the victim's network. Attacker hosts a website *https://attacksite.com* and hosts a DNS service to resolve the domain *attacksite.com* whose TTL is set to 0.

For simplicity Client has an SMTP server running on his localhost.

The attack :

1. Victim connects to *https://attacksite.com,* using a client application (browser/curl or nc). The TLS session is established and **Session ID** is obtained

2. Attacker redirects user to *https://attacksite.com:25* i.e to SMTP port. Since the DNS entry havin TLS set to 0 the client application queries for the IP address of the *https://attacksite.com:25.* To which DNS server responds with IP of localhost 127.0.0.1
3. The attacker-controlled server sends continuous connection requests with the same **Session ID** to maintain a deferred-persistent session over which it sends SSRF payload to the SMTP port on localhost and tricks it to send a spam mail.

Joshua demonstrated attacks on **memcache** service running on the client side to insert value of his choice in the cache and also to execute remote code.

Impact

The successful SSRF attack ability is attributed to TLS protocol's feature to resume TLS sessions over multiple TCP requests using **Session ID** or **Session Tickets**. The attacker can abue this feature for information disclosure, sending spam or phishing mails from victims side, remote code execution or carrying out DDoS attacks.

Mitigation

Author suggested the Session Resumption feature of TLS should be relooked by the community to avoid such attacks. Change in the TLS specification is a long and complex process mean while Null Cryptography community recommends the regular mitigations that are practiced to avoid SSRF like

1. Validate input data for redirection to localhosts or internal servers
2. Validation of protocol commands to avoid any cross protocol exploitation like exploitation of SMTP over HTTP(S)
3. Block unauthorized access to any services

Reference

1. Author's presentation https://www.youtube.com/watch?v=qGpAJxfADjo
2. Author'sGithub https://github.com/jmdx/TLS-poison
3. Other references:
   - https://portswigger.net/web-security/ssrf
   - https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html
   - https://en.wikipedia.org/wiki/DNS_rebinding

6.5.14 CVE-2020-11651,52 Allows Unauthorized Access to SaltStack Master Server

Introduction

An issue was discovered in SaltStack Salt before 2019.2.4 and 3000 before 3000.2. The salt-master process ClearFuncs class does not properly validate method calls. This

allows a remote user to access root keys, leading to access of sensitive methods without authentication.

This flaw was used to attack DigiCert CA and attackers could get its Certificate Transparency (CT) Log2's keys used to sign SCT (Signed Certificate Timestamp)

CVSS Score: 9.8 / Critical (CVE-2020-11651), 6.5 / Medium (CVE-2020-11652)

Description

Salt is an open-source server configuration, update management & monitoring tool. It is sed to monitor the state of servers at data centers and remote networks. Servers to be monitored are agents called minions, which report the server status to the Master. Master can also push updates, and issue commands to minions.

In May 2020, F-Secure security researchers disclosed two vulnerabilities in Salt (CVE-2020-11651 and CVE-2020-11652) that could allow remote attackers to execute commands as root on "master" and connected minions. The most severe of the bugs has a CVSS score of 10.

These vulnerabilities were used to attack Digicert CA server and compromise its time stamp signing keys.

These vulnerabilities are not directly related to cryptography but it underscores the importance of securing the keys and secret parameters. It's obvious that a very well & securely implemented cryptography fails if keys are not secured appropriately.

Exploitation

As stated by Trend Micro's research team in the research blog; In CVE-2020-11651, the salt master process, ClearFuncs(), does not validate method calls properly. This class exposes two methods:

- **_prep_auth_info() method**, which returns the user token (Root Key). Attackers can use the token to bypass authentication, resulting in remote command execution within the context of the Salt process on both the master and minions.
- **_send_pub() method**, which can be used to queue messages directly on the master publish server. Such messages can be used to trigger minions to run arbitrary commands as root.

This vulnerability allows access to some methods without requiring authentication. This then can be used to retrieve user tokens from the salt master and run commands on salt master and salt minions.

CVE-2020-11652 is also related to the salt master process ClearFuncs(); this time, it allows access to some methods that improperly sanitize paths.

Together CVE-2020-11651 & CVE-2020-11652 allows complete takeover of the Salt Master server.

Impact

Total takeover of infrastructure is possible when root keys are exposed as in the case of CVE-2020-11651. It was rated Critical with CVSS score of 9.8

Mitigation

At Null cryptography group we recommend the following best practices for administrators and developers to make access to keys and secret data difficult

1. Securely store root keys or sensitive data in secure vault like HSM etc. which can be accessed by authorized users and process only
2. While programming make sure you are not exposing keys, passwords or secret parameters to clients/remote users. In most cases it is never required.
3. Storing keys in DER format and with cryptic names do not secure the keys from compromise but can definitely offer some help to delay the abuse of the keys by the attacker.
4. Administrators and responders should quickly revoke the resources signed by the compromised keys.
5. inform other competent authorities, CAs to distrust the resources singed by compromised key
6. Sanitize your server before refreshing the signing keys.

Reference

1. https://www.tenable.com/blog/cve-2020-11651-cve-2020-11652-critical-salt-framework-vulnerabilities-exploited-in-the-wild
2. https://www.trendmicro.com/vinfo/in/security/news/vulnerabilities-and-exploits/coinminers-exploit-saltstack-vulnerabilities-cve-2020-11651-and-cve-2020-11652
3. https://www.securityweek.com/recent-salt-vulnerabilities-exploited-hack-lineageos-ghost-digicert-servers