null – The Open Security Community

Learning from the Past: Cryptography Vulnerabilities Exploited in 2019

White Paper

Contents

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.

Acknowledgement

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

- 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/)
- Ajit Hatti (https://www.linkedin.com/in/ajithatti/)

Abstract

Every second, tons of data gets transmitted, stored, and processed over the internet across billions of nodes. Cryptography is at the foundation of Information security to keep this data secure at all three stages of the data life cycle. Recently in the advancement of technology and adoption of technology like Cloud, IoT and Blockchain have increased the usage and importance of cryptography.

Going beyond the conventional use cases of cryptography, it is also the genesis of disruptive technologies like Blockchain & Crypto-currency. As the usage of cryptography has increased tremendously, a lot of research has been carried out to break cryptography implementation to make it more robust and strong. The main aim of this paper is to provide a brief about all the cryptographic related vulnerabilities exploited in 2019.

Motivation

Cryptography being a foundational discipline in cybersecurity, the discussion around its vulnerabilities, good practices standardization in practices is left to the cryptographer who are 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 at wider realm to adopt and practise the affinity for cryptography & its secure implementation.

Leveraging the forum of NULL open security community, we attempt to analyse the cryptographic vulnerabilities of 2019 which have been impacting the year 2020 and we expect them to continue to do the same until we seriously pay attention to better practices and procedure when it comes to cryptography implementation and securing the cryptographic resources.

1. Overview

We have collected and organized most of the cryptography implementation related vulnerabilities which are exploited in 2019. We have classified these vulnerabilities into below mentioned five major categories :

1. Side Channel
2. Certificate Related Flaws
3. Buffer Overflow
4. Padding Oracle
5. Others.

For all the vulnerabilities we have provided the following details :

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

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

2. Vulnerabilities Detail

2.1 Side Channel related attacks

2.1.1 RAMBleed: Reading Bits in Memory WithoutAccessing Them

Introduction

RAMBleed exploit is based on the Row-Hammer vulnerability where repeated access to the rows in DRAM can lead to bit flips in the adjacent rows (not only the direct neighbors), even if these neighboring rows are not accessed directly. RAMBleed uses Row-Hammer to read the bits in the victim's memory. This is an attack on confidentiality where the secret data in the security domains can be read without accessing those security domains/ processes/ memory directly.

Description

The Rowhammer bug is a reliability issue in DRAM cells that can enable an unprivileged adversary to flip the values of bits in neighboring rows on the memory module. Using Rowhammer exploit,unprivileged attacker can exploit the data dependence between Rowhammer-induced bit flips and the bits in nearby rows to deduce these bits, including values belonging to other processes,thus it is a Read side channel attack and it is not only a threat to the integrity of the data but also to the confidentiality of the data as well.

Exploitation

RSA-2048-bit key can be retrieved by using RAMBleed from OpenSSH server using only user level permissions.

Exploitation mechanism:

Row activation page	**secret**
Unused bit	**Sampling page**
Row activation page	**secret**

1. Attacker uses Row activation pages for hammering the Sampling page that contains a flippable bit, secret page contains the secret data of the victim.

2. Here, if the value in the secret page is 0 the layout of the flippable bit in the Sampling page would be 0-1-0. Now, by using Row-Hammer, constantly hammering the 2 rows above and below, Row-Hammer induces a bit in the sampling page and thus, the flippable bit gets flipped. So, configuration becomes 0-0-0. This is how an attacker learns if 0-0-0 configuration has occurred, means secret value is 0 else secret value is 1, deducing the secret value without accessing the secret value.

- As this is a side channel attack, no antivirus solution in the market can detect or remediate this attack.
- CVE-2019-0174 can be referred for this vulnerability.
- No ECC (Error correcting codes) can stop this attack.

Impact

Successful exploitation of this vulnerability could lead to disclosure of sensitive information via local access.

Mitigation

- Users can mitigate their risk by upgrading their memory to DDR4 with targeted row refresh (TRR) enabled.
- Though Row-hammer induced bit flips have been demonstrated on TRR, it is harder to accomplish in practice.

2.1.2 Found timing vulnerabilities in the DSA and ECDSA implementations of the RSA BSAFE library

Introduction

Exploitation of Side channel timing attack against (EC) DSA in RSA BSAFE results in revealing of private key and breaking of digital signature algorithm.

Description

Using timing attack, analyzing time factor for various signatures, it is able to deduce the prime factor values or secret values in order to recover the private key. The technique used to deduce or recover the private key is the Lattice attack, which focuses on the Digital signature algorithms.

Exploitation

- Requesting Digital signatures formed using (EC) DSA repetitively and measuring the time required each time for the processing is used to deduce the private key used in the transaction
- Information Exposure Through Timing Discrepancy – CVE-2019-3739 can be referred

Impact

- RSA BSAFE Crypto-J versions prior to 6.2.5
- RSA BSAFE SSL-J, all currently supported versions where 6.2.4.1 is the most recent release
- RSA BSAFE Cert-J, all currently supported versions where 6.2.4 is the most recent release

Mitigation

- It is necessary to use RSA BSAFE Crypto-J version 6.2.5, which contains a fix for the vulnerability.

References

- https://blog.intothesymmetry.com/2019/08/side-channel-timing-attacks-against.html

2.1.3 Elliptic curve implementations vulnerable to Minerva timing attack

Introduction

Elliptic curve implementations vulnerable to Minerva timing attack Researchers from Masaryk University have presented a timing attack called Minerva. The attack exploits side-channel vulnerabilities in implementations of elliptic curve signatures, primarily ECDSA. These attacks allow for practical recovery of the long-term private key. The primary targets of the attack were cryptographic chips with certifications from FIPS and Common Criteria, but several open-source cryptographic software libraries were also affected. The FIPS 140-2 certification scheme specifically does not require side-channel resistance to be tested by the lab performing the assessment. So even though the FIPS security targets of the aforementioned cards specify resistance against side-channel attacks, no such testing had to be performed. The original Common Criteria certificate DCSSI-CC-2009/11 that introduced the vulnerable functionality did so by stating the functionality is explicitly not protected against SPA/DPA attacks and should not be used on secure data

Description

The vulnerable implementations of ECDSA leak the bit-length of the scalar during scalar multiplication on an elliptic curve. This leakage might seem minuscule as the bit-length presents a very small amount of

Information present in the scalar. However, in the case of ECDSA signature generation, the leaked bit-length of the random nonce is enough for full recovery of the private key used after observing a few hundreds to a few thousands of signatures on known messages, due to the application of lattice techniques.

Vulnerability is present in the underlying firmware, which is inaccessible to the user/administrator of the device. However, the firmware might be updateable by the manufacturer. In the case of a vulnerable library, updating it to the newest version should fix it, as most libraries we notified fixed the issue and released a new version.

Following CVE are released :

- CVE-2019-15809 : Vulnerability in Athena-based cards.
- CVE-2019-13627 : Vulnerability in libgcrypt.
- CVE-2019-13628 : Vulnerability in wolfSSL/wolfCrypt.
- CVE-2019-13629 : Vulnerability in MatrixSSL.
- CVE-2019-2894 : Vulnerability in SunEC/OpenJDK/Oracle JDK.
- CVE-2019-14318 : Vulnerability in Crypto++.

Exploitation

Minerva attack is a lattice attack on the timing leakage of the bit-length of nonces used in ECDSA and other similar signature algorithms. The vulnerable devices and libraries trivially leak the bit-length of the scalar used in scalar multiplication in ECDH, ECDSA and key generation. The leakage is insignificant for ECDH and key generation as only the bit-length of the private key is leaked, which represents a small amount of always the same information about the private key.

However, in the case of ECDSA or other signature schemes with random nonces, the bit-length of the random nonces is leaked. This is much more significant as each signature then presents new usable information on the private key. The way this information is used to recover the private key is via first converting the problem to an instance of the Hidden Number Problem, which we describe and solving it via lattice reduction techniques.

The attacker needs to be able to measure the duration of hundreds to thousands of signing operations of known messages. The less noise in the measurement is present, the less signatures the attacker needs. The computation of the private key is then a matter of seconds or minutes. The attack is exploitable locally if the runtime of signing operations performed by an affected device or library is measurable, locally the amount of noise is not enough to stop the attack.

The Minerva proof-of-concept contains code that exploits the vulnerability against a JavaCard with a target applet, or against targets using several vulnerable libraries. The Minerva CPLC checker uses the Card Production Life Cycle (CPLC) data present on cards under the GlobalPlatform standard to identify vulnerable cards based on a list of known or suspected vulnerable devices. The Minerva tester is a testing tool for JavaCards which uses ECDH to assess the presence of timing leakage of bit-length in scalar-multiplication.

Impact

Affected devices and libraries:

- Devices - Athena IDProtect
- Libraries - SunEC/OpenJDK/OracleJDK, Libgcrypt, MatrixSSL, WolfSSL/WolfCrypt, Crypto++

Mitigation

Athena IDProtect cards no longer exist, as it was bought by NXP in 2015. NXP Semiconductors confirmed the existence of the vulnerability on 19.04.2019 and stated that they had migrated the former Athena IDProtect product line to their hardware shortly after the acquisition, which effectively mitigated the vulnerability in newer products based on the IDProtect line, long before our discovery, as it was present in the underlying cryptographic library which was replaced along with the hardware. Furthermore, NXP discontinued the former Athena IDProtect line of products a few years after the acquisition.

Patches are released for vulnerabilities in libraries of Libgcrypt, wolfSSL, Crypto++ and SunEC. Vulnerability in MatrixSSL & public jdk repository remains unpatched as of 02.10.2019.

References

1. Elliptic curve implementations vulnerable to Minerva timing attack https://www.feistyduck.com/bulletproof-tls-newsletter/issue_58_elliptic_curve_implementations_vulnerable_to_minerva_timing_attack
2. https://minerva.crocs.fi.muni.cz/

2.1.4 The TPM-Fail attack shows timing side-channels in elliptic curve operations of TPM chips. The vulnerable chips had certifications from Common Criteria and FIPS that failed to detect these weaknesses.

Introduction

Trusted Platform Module (TPM) serves as a root of trust for the operating system. TPM chips are also used in other computing devices such as cellphones and embedded devices. TPM is supposed to protect our security keys from malicious adversaries like malware and rootkits. The TPM-Fail attack shows timing side-channels in elliptic curve operations of TPM chips. The vulnerable chips had certifications from Common Criteria and FIPS that failed to detect these weaknesses.

Description

Researchers Daniel Moghimi, Thomas Eisenbarth et al. discovered timing leakage on Intel firmware-based TPM (fTPM) as well as in STMicroelectronics TPM chip. Both exhibit secret-dependent execution times during cryptographic signature generation. While the key should remain safely inside the TPM hardware, leakage of this information allows an attacker to recover 256-bit private keys from digital signature schemes based on elliptic curves.

Common Criteria certification is obtained via independent evaluation and is meant to provide security assurance including resistance to physical side-channel and timing attacks. The STMicroelectronics TPM chip is Common Criteria certified at EAL4+ for the TPM protection profiles and FIPS 140-2 certified at level 2, while the Intel TPM is certified according to FIPS 140-2. However, the certification has failed to protect the product against an attack that is considered by the protection profile.

CVE-2019-11090 for Intel fTPM vulnerabilities and CVE-2019-16863 for STMicroelectronics TPM chip are released.

Exploitation

A hacker can use these vulnerabilities to forge digital signatures. Lattice techniques were applied to retrieve 256-bit private keys for ECDSA, ECDAA and ECShnoor.

If your operating system or any of the applications on your computer use the TPM to issue such digital signatures, the private signing key used for signature generation can be compromised. Compromised signing keys can be used to forge signatures for bypassing Authentication, tampering the OS, and other bad things depending on what the digital signatures are used for.

Attacks are practical. Local adversaries can recover the ECDSA key from Intel fTPM in 4-20 minutes depending on the access level. Attacks can be performed remotely on fast networks, by recovering the authentication key or digital signature of a StrongSwan IpsecVPN server in 5 hours.

Impact

Any of computing systems (laptop, desktop, server workstations, tablet etc) that use Intel fTPM firmware, STMicroelectronics TPM chip

Mitigation

- Intel fTPM firmware update got released on 11/12/2019 to fix the reported vulnerabilities
- STMicroelectronics provided a new TPM firmware to be loaded on the same TPM device that is resistant against TPM-FAIL

References

- TPM-Fail: TPM meets Timing and Lattice Attacks
  https://tpm.fail/
  https://www.usenix.org/system/files/sec20_slides_moghimi-tpm.pdf
- Remote Timing Attacks on TPMs, AKA TPM-Fail
  https://i.blackhat.com/USA-20/Thursday/us-20-Moghimi-Remote-Timing-Attacks-On-TPMs-AKA-TPM-Fail.pdf

2.1.5 TPM-FAIL attack on cryptographic co-processor : Timing side-channels on TPM chips

Introduction

Trusted Platform Module (TPM) serves as a root of trust for the operating system. TPM chips are also used in other computing devices such as cellphones and embedded devices. TPM is supposed to protect our security keys from malicious adversaries like malware and rootkits. The TPM-Fail attack shows timing side-channels in elliptic curve operations of TPM chips. The vulnerable chips had certifications from Common Criteria and FIPS that failed to detect these weaknesses.

Description

Researchers Daniel Moghimi, Thomas Eisenbarth et al. discovered timing leakage on Intel firmware-based TPM (fTPM) as well as in STMicroelectronics TPM chip. Both exhibit secret-dependent execution times during cryptographic signature generation. While the key should remain safely inside the TPM hardware, leakage of this information allows an attacker to recover 256-bit private keys from digital signature schemes based on elliptic curves.

Common Criteria certification is obtained via independent evaluation and is meant to provide security assurance including resistance to physical side-channel and timing attacks. The STMicroelectronics TPM chip is Common Criteria certified at EAL4+ for the TPM protection profiles and FIPS 140-2 certified at level 2, while the Intel TPM is certified according to FIPS 140-2. However, the certification has failed to protect the product against an attack that is considered by the protection profile.

CVE-2019-11090 for Intel fTPM vulnerabilities and CVE-2019-16863 for STMicroelectronics TPM chip are released.

Exploitation

A hacker can use these vulnerabilities to forge digital signatures. Lattice techniques were applied to retrieve 256-bit private keys for ECDSA, ECDAA and ECShnoor.

If your operating system or any of the applications on your computer use the TPM to issue such digital signatures, the private signing key used for signature generation can be compromised. Compromised signing keys can be used to forge signatures for bypassing Authentication, tampering the OS, and other bad things depending on what the digital signatures are used for.

Attacks are practical. Local adversaries can recover the ECDSA key from Intel fTPM in 4-20 minutes depending on the access level. Attacks can be performed remotely on fast networks, by recovering the authentication key or digital signature of a StrongSwan IpsecVPN server in 5 hours.

Impact

Affected products: Any of computing systems (laptop, desktop, server workstations, tablet etc) that use Intel fTPM firmware, STMicroelectronics TPM chip.

Mitigation

- Intel fTPM firmware update got released on 11/12/2019 to fix the reported vulnerabilities
- STMicroelectronics provided a new TPM firmware to be loaded on the same TPM device that is resistant against TPM-FAIL

References

- TPM-Fail: TPM meets Timing and Lattice Attacks
  https://tpm.fail/
  https://www.usenix.org/system/files/sec20_slides_moghimi-tpm.pdf

- Remote Timing Attacks on TPMs, AKA TPM-Fail
  https://i.blackhat.com/USA-20/Thursday/us-20-Moghimi-Remote-Timing-Attacks-On-TPMs-AKA-TPM-Fail.pdf

2.2 Certificates related flaws

2.2.1 Insecure TLS Proxy leads to MITM (Man in the middle) attack in content filtering / parental control appliances

Introduction

This issue describes trust issues for content filtering products which use TLS proxy but are failed to maintain adequate level of security by trusting the dummy Root Agency certificate or failed in doing proper certificate chain validation as part of their TLS proxy, exposing users to trivial man-in-the-middle attacks on HTTPS traffic.

Description

Many enterprise grade network appliances host a TLS proxy to facilitate interception of TLS – protected traffic for various purposes, including scanning, phishing detection, content filtering, preventing data exfiltration etc. When deployed TLS proxy acts as a client to a web server and acts as a server to a client which is requesting. However, if the level of security for the proxy is not up to the benchmark at least the level of the same level as modern web browsers and properly configured TLS servers, then it increases the attack surface for all the proxied clients serving the network appliance.

Exploitation

Below is a thread model to depict how Man in the middle attack can be exploited by various threat actors :

1. ***External hacker***
   - Impersonate any web server by performing a MITM attack on a network appliance that is not validating a certificate validation.
   - Even if the validation process is perfect, an attacker can still impersonate any web server, if the appliance uses a pre generated root certificate and accepts external site certificates signed by its own root key.
   - Attackers who compromise the network appliance itself with non-root privileges can require the private key if the key is not properly protected. Without proper ACLs on private keys.

2. ***Internal hacker***
   - Internal hackers can take advantage of the network sniffer with promiscuous mode and can get access to the raw traffic from the connections between network appliances and client

Real life scenarios:

Affected appliances having vulnerability due to insecure TLS proxy:

- *Net nanny*
  - Popular parental control application to restrict and filter bad content from webpages visited by children.
  - Tested versions include 7.2.4.2, 7.2.6.0,7.2.8.0 however earlier versions could be possibly affected.
- *Untangle Firewall*
  - Network appliance to control and filter traffic in enterprise networks.
  - Tested versions include 13.0 and 13.2.

Steps and resources for practical exploitation of above appliances:

- Obtains a copy of makecert.exe as part of the Microsoft Windows SDK
- Extracts the private key for the Root Agency certificate (located in a resource, easy and one-time effort)
- Targets a victim that uses Net Nanny on a Windows computer or Untangle NG Firewall, and intercepts its network traffic
- Intercepts TLS connections and provides certificates signed by Root Agency's private key retrieved in point #2.

If the warning doesn’t come from the browser for this link, then it is vulnerable.

- https://rootagency-test.encs.concordia.ca/

Below diagram depicts how TLS proxy intercepts TLS traffic between web server and client:

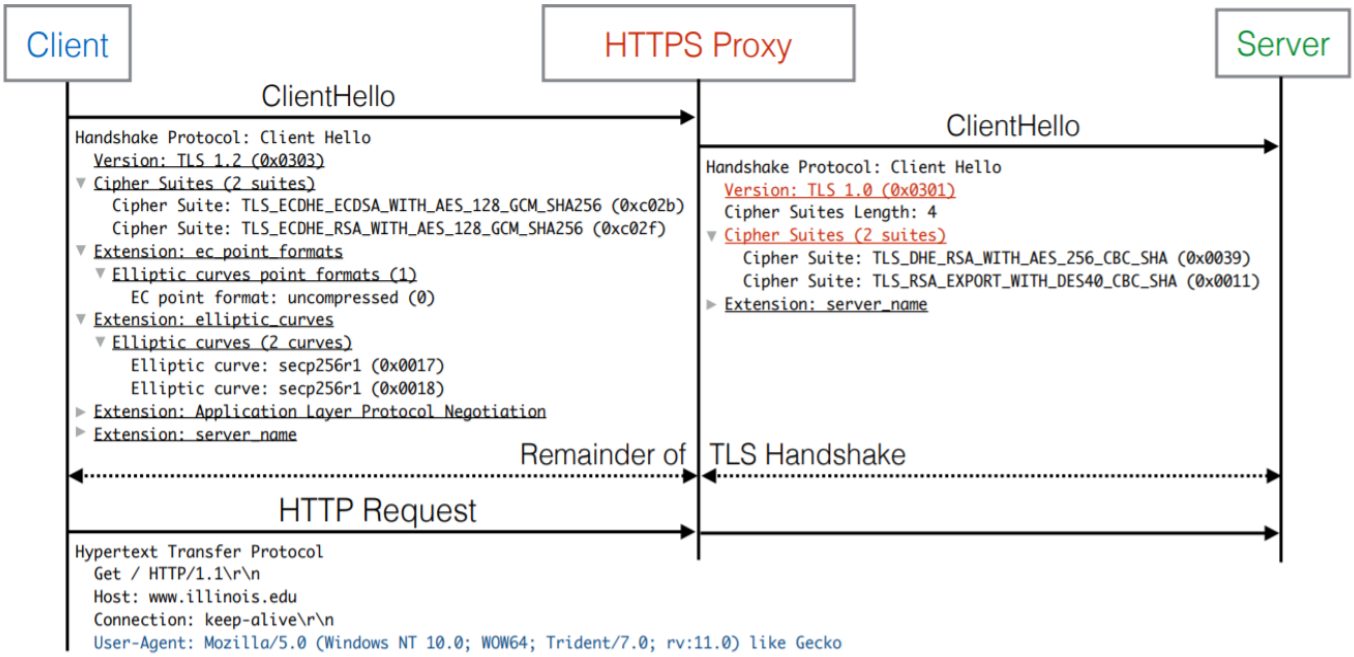


Figure: TLS proxy working intercepting TLS connection between web server and client

Impact

Due to insufficient security maintained at the proxy level, though the server or client is maintaining high level of security , it is possible to do MITM attack, and steal sensitive information like PII(Personally Identifiable data), Passwords and more adversely, hijacking a session leading to session hijacking.

Mitigation

- TLS Version, key length, and signature algorithms should be mirrored in TLS proxy to avoid misleading clients regarding TLS security parameters used in the proxy to external web server connection.
- The list of cipher suites offered by the client should ideally be mirrored to the servers TLS connection. Domains with EV certificates should not be downgraded to DV certificates, by exempting them from the interception process.
- TLS proxies and any associated libraries must be up to date and patched against known TLS attacks and vulnerabilities. (BEAST, CRIME, FREAK, Logjam, TLS insecure renegotiation)

- As TLS proxy will be having responsibility for protecting the clients by performing proper certificate validation on behalf of them, browsers will be only exposed to proxy issued certificates. A less stringent certificate validation would result in a consequence of MITM, and downgrading attacks. TLS chain validation should be properly validated against flaws such as :
  - Untrusted issuers
  - Mismatched signatures
  - Wrong common names
  - Revoked and expired certificates
  - Short key certificates
  - Deprecated signature algorithms
- Should also recognize their own root certificates provided by web server(which would never occur_ and block them
- Proxies trusted root store must not contain short Ky, expired or untrusted certificates

References

1. https://elie.net/static/files/the-security-impact-of-https-interception/the-security-impact-of-https-interception-paper.pdf
2. https://madiba.encs.concordia.ca/~x_decarn/rootagency.html

2.2.2 1300 incidents where CAs have broken rules from the CA/Browser Forum and from browser root programs

Introduction

In a comprehensive research paper published by Nicolas Serrano, Hilda Hadan & L Jean Camp From School of Informatics,Computing & EngineeringIndiana UniversityBloomington, 1300 instances of violation of basic operating guidelines recommended by CA/Browser Forum are studied.

Technical Details

Various technical issues and validation laps were seen across all the certification authorities in their management & technical process.

1. Weak Kyes - 512/1024 bit key certificates were issued
2. CA/RA/Reseller got hacked - due to use of simple passwords
3. CAA Misissuance - certificates issued to unregistered domains
4. Erroneous/Misleading/Late/LackingAudit report - delayed revocation of compromised certificates
5. Fields in certificates not compliant to BR - integrity issues in the certificates
6. Non-BR-compliant or problematicOCSP responder or CRL
7. Possible issuance of rogue certificates
8. Repeated/Lacking appropriate entropy Serial Numbers
9. Rogue certificate
10. Undisclosed SubCA
11. Use of SHA-1/MD5 hashing algorithm

Apart for the above issues the following issues were observed with some CAs

1. Backdating SHA-1 certificates•Certificates for malicious domains
2. Charging for compromised cert revocation (HeartBleed)
3. CPS non-compliance•Debian OpenSSL Vulnerability
4. Delayed certificate revocation
5. Digital certificate for non-existent domain/Bad informa-tion of requester
6. MITM attempt
7. No BR self-assessment
8. No disclosing another CA purchase

9. No test website for CA
10. Non-acceptable requester validation
11. Not allowed ECC usage
12. Registering competitor trademark
13. Self revocation
14. Timestamp certificate from root
15. Un-revoking certificates
16. Validity greater than 825 days

Exploitation

The lack of appropriate procedures and adherence to best practices can be exploited by advertisers to

1. Generate & distribute untrusted certificates
2. Generate certificates with weaker keys (Weak modulus attack)
3. Adversaries can steal keys of widely trusted organisations and (ab)use them to sign malwares
4. By stealing the private keys adversaires can snoop on user communications trusting the corresponding certificates

Impact

Its difficult to assess the impact of 1300 incidents of Irregularities shown by Certifying Authorities in issuing certificates. The very nature of certificate manipulation attacks is to thoroughly conceal the fraud underway.
What can be seen is many Certifying Authorities and their subsidiaries were mistrusted and forced to close their brands.

List of Untrusted CA's:

CA	Major Incidents
Comodo	Lack of validation in certificate issuance and unethical practices
WoSign	Various irregularities including - if a certificate requester was in control of a subdomain, then he could have a certificate for the entire domain
Symantec	Chronic failures in organization, technical, and purposeful business driven errors
VeriSign	Multiple problems & bad practices exhibited through other CAs it owned like Equifax, GeoTrust & RapidSSL
DigiCert	Issued certificates with weak keys (512 bits) & lack of the key usage definition. Which were then duplicated and used to sign malware.
CNNIC	MITM scenario involved the root certificate of Chinese certificate authority and domain name registry, CNNIC, and the intermediate CA was MCS Holdings. Google identified that the private key of the intermediate CA was misused to create rogue certificates for Google's domains
DigiNotar	Servers were compromised and several rogue certificates were issued
India CCA	Forgery Google, Yahoo''s certificates, maybe with MITM goals using its intermediate CA belonging to the NIC
StartCom	Unfair business practices. Issued free certificates but charged for revocation amid Heartbleed crises

Mitigation

Certification Authorities must comply with standard operational procedures led by the CA Browser forum. Important points to be noted

1. Follow high level of compliance controls in the process of certificate issuance and regular audits, baseline requirements & strong encryption and hashing algorithms
2. Manage the software vulnerabilities/Bugs in the ecosystem
3. Thoroughly enforce certificate validation rules
4. Thorough checks on the intents and business models of intermediate Certifying or registration authorities
5. Thorough testing & proper security controls in place

2.2.3 Certificates related flaws -Certificate Authority Certinomis removed from Firefox browser

Introduction

Andrew Ayer in April 2019 reported 14 certificates issued by Certinomis to unregistered domains. Mozilla has captured conversation between its members & certinomis official including the reference to previous issues with Certinomis in a Wiki page.

Considering the multiple incidents of irregularities and lack of response & ownership demonstrated by Certinomis, Mozilla decided to distrust Certinomis completely and removed its root certificate from Mozilla root store.

Details

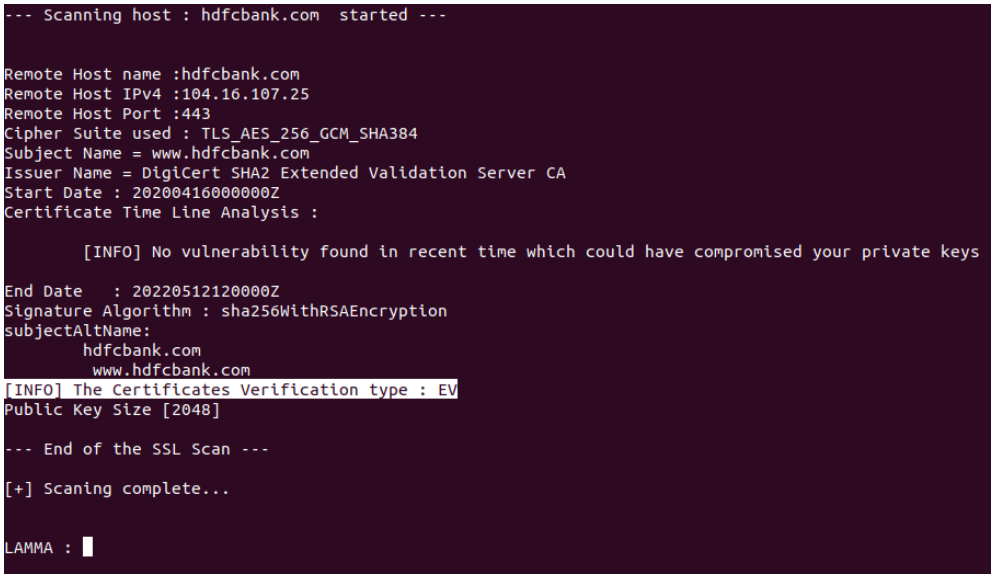
X509 certificates are issues in 4 categories

- **Domain Validation (DV):** Issued to a requester for a domain on which he can prove control. Cheap (free), quick & automated issuance of certificates.

Lamma tool shows the Domain Validation (DV) certificate of null.co.in

- **Organization Validation (OV) :** Issued to a requester after a verification of the identity of the organization (e.g. a business, nonprofit, or government organization).
- **Individual Validation (IV) :** Issued to individuals after verification of individual's identity.

- **Extended Validation (EV)**: Issued to an organisation after through verification. It assures involvement of higher standards than OV. Previously browsers used to treat such certificates with special visual cues to denote higher trust.



Lamma tool shows the Extended Validation (EV) certificate of hdfcbank.com

In the current case, Certinomis failed to provide the reasoning or validation process used to issue certificates for unregistered domains. This demonstrates lack of procedural standards, commitment to quality and misplaced intent.

Exploitation

The certificates issued by Certinomis to un-registered organisations are not seen in any publicly known incidents, however such certificates can be used by adversaries to exploit users' trust and have plausible deniability & traceability.

Impact

The intent of issuing such a certificate was not clear. Also the impact of such certificates on the end user is not known. The (yet) another example was set by untrusting certinomis.

Mitigation

CAs need to issue certificates after appropriate validation as applicable for the category requested by the subject.

2.2.4 Extended validation certificates have “Default City” in locality

Introduction

Extended validation certificates have detailed information about the server publishing the certificate so as to prove the control on the Domain name server (DNS) and discloses more valid information about the organization to increase the trust. However, the creation of domain validated certificates should be in accordance with specified Business requirements. Some of the extended validation certificates have breached this criteria by specifying “Default City” in the subject address line.

Description

Extended validation certificates have more information about the server publishing the certificate , it proves control on Domain Name server(DNS). However, some of the extended validation certificates are not in line with the Business Requirements and specified “Default City” in the locality that is subject address line making it violating the compliance breaker.

Exploitation

According to BRs, section 3.2.2.1 and 7.1.4.2.2.e. states, certificates need to specify the locality that is the subject address line. However, Default City is not matching the criteria.

The following links include certificates which are revoked and were having locality as Default city.

misissued.com

https://misissued.com/batch/51/

Impact

The certificates were not compliant and were not complying fully to the defined standard per mentioned for Extended validation certificates.

Use of these certificates would lead to compliance issues. However, most of the certificates in the list are revoked by the respective Certificate Authorities.

Mitigation

- Above vulnerability is more towards compliance of the extended validation certificates with the respective business requirement sections.
- Party which issues the certificate should not incur such vulnerabilities as this is more human agent related (typographical) vulnerability.
- Part which uses such certificates should verify compliance before using such certificates.

References

https://groups.google.com/g/mozilla.dev.security.policy/c/1oReSOPCNy0?pli=1

2.2.5 Amazon music app is vulnerable to serious compromise of private key vulnerability

Introduction

Amazon music application is vulnerable to serious compromise of the private key used to communicate with the amazon local music application web server. Amazon music application used Security through Obscurity approach by encrypting RSA Private key used to communicate and establish a communication with the amazon music application web server. The purpose of a private key is to keep it private always failing to which the CA needs to revoke the private key.

Description

- Private key for the Amazon music application was compromised leading to the result that the application can be launched like a zoom app from web pages without any interaction with users.
- The way it works is accepting HTTP requests via the local web server . This bypasses the built in safety net in browser that takes confirmation from users before launching the application.
- Steps to compromise the private key of the app:
  - Using disassembler to debug amazon music app binaries to extract AES symmetric key
  - Using symmetric key decrypt the certificate and private key of the certificate
  - Once the private key is compromised, Any attacker would then intercept the communication or sign fake certificates and use it against the victim machine or connect to the amazon app server pretending to be a legitimate entity.

Exploitation

1. Install amazon music app in a sandboxed environment or clean virtual machine.
2. Use any command to show live connections on the ports, e.g. netstat
   a. As depicted below, amazon music helper listens on port 18000 and then it is an observation that it binds to any other interface and not just the loopback interface

```
~ $ lsof -i -P | grep LISTEN
Amazon    321 privileged   22u  IPv4 0x103232efe6013a0f      0t0  TCP *:18800 (LISTEN)
~ $
~ $ ps -o pid,command -p 321
  PID COMMAND
  321 /Applications/Amazon Music.app/Contents/MacOS/Amazon Music Helper
~ $
```

3. Use cURL to verify that the local web browser presents a valid and publicly trusted certificate to establish valid HTTPS connection.
4. As it is a binding HTTPS connection, it means a private key is present in the binary.
5. Use a disassembler to extract the private key.

```
amazon-music $ openssl enc -aes-256-cbc -nosalt -d -in key.der.enc \
> -K "fb4098e00b3fb0562a625d96599b2be6a3876ccfc21908c1cb173850745d1cd5" \
> -iv "3ac0660ee766394f65a6eb26ae3dbf84" |
> openssl rsa -inform der -outform pem
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAjQVtI25Ix9WroWPSaXFV3JFS6cEi4Ux8c6B1Y7CDRc7O8Fcr
57hUlTCeCFxxXAgOllVOn3SQIZT8OSiOFoEvrdc2Xb/WHZJaTZnc89hM2jC9DB/p
ateiXWRS2ZGJzt3gqWWRVVWM2QEAt3Zr3UV68CbCez6mCd+1uI0EtJP7V47bNSL0
uLEpnjeE0NVM6HsYmwdfljmcZlzWoEvEFCAQkIddPuhLLowbLqTlTGuUWIGiWk1Q
Ctk531w9Ddg2wlJ3P+vyXtjR11vaS9k57V66obaF7aaEGXgezqHY5qVjjanvdLvj
VV0sX73u+MmWwh6KLnvJXRcuJWolDLie6nDQlQIDAQABAoIBAE5baIQaYGhhSmd3
rs+WAO549UgbX8UuGaypyPBmgNqGCigzV5r+qkjnZbghnX9Unr5bkj85pmhyfx9H
cIQCrjp67aNz14ZA7tpVlxcK5dEK6dBV3v3V6UrZVsu7eYYvdQjK/dEboxJk40AN
leuyEphkAQY4DkxIOJHeEXJBE5RXYjkQW93KUHuRBpV/JLx+Ky1g0hycCkmW+s0q
whB3Ae2xcKNiIWjOLCk+PCT7Cn1Egbz4s6MoRkoF+A7VoTHCIsG2OodT+FlWg7Ec
cMIjW00ZVqiGL+ne175JdpOO6HAZL7XTChk6isMbG2CmNdXQO/VVR/rgJQ1WVd3E
U2Z+VEECgYEAwWF7MEHON9IUn/68o7fabGkZksGFEuvYIDJ27hSBuP8djb005l6n
MFBevSzooCcE2/ETCEQj7RH64vIc/nMz4AXYrj82bUnG9mDRQzVH15yDr/n/xFXW
zn3t5jaXHu8mlMNJOIQG+yBsc/0tAoBeXsZNTrxqt+PrFDd3HSzgPHECgYEAuq+J
rBVEA4j9EGAVpn//burHD50Pp4x+DzXz1HNIpIrYOsQCj5VfL0Zi+GMstnP8H+24
```

Impact

Compromising the private key for a certificate for amazon music app is an example of insufficient security and thus the vendor is failed to maintain sufficient security to keep the private key as private. Henceforth, amazon has revoked the certificate. However, compromise of private key would lead to eavesdropping of the traffic between client and the amazon music app server.

Mitigation

This vulnerability doesn't exist any more as the certificate is revoked by Amazon.

References

- https://koen.io/2019/07/26/underscoring-the-private-in-private-key/

2.2.6 Mozilla & Google distrusts DarkMatter an intermediate CA for involvement in Project Raven

Introduction

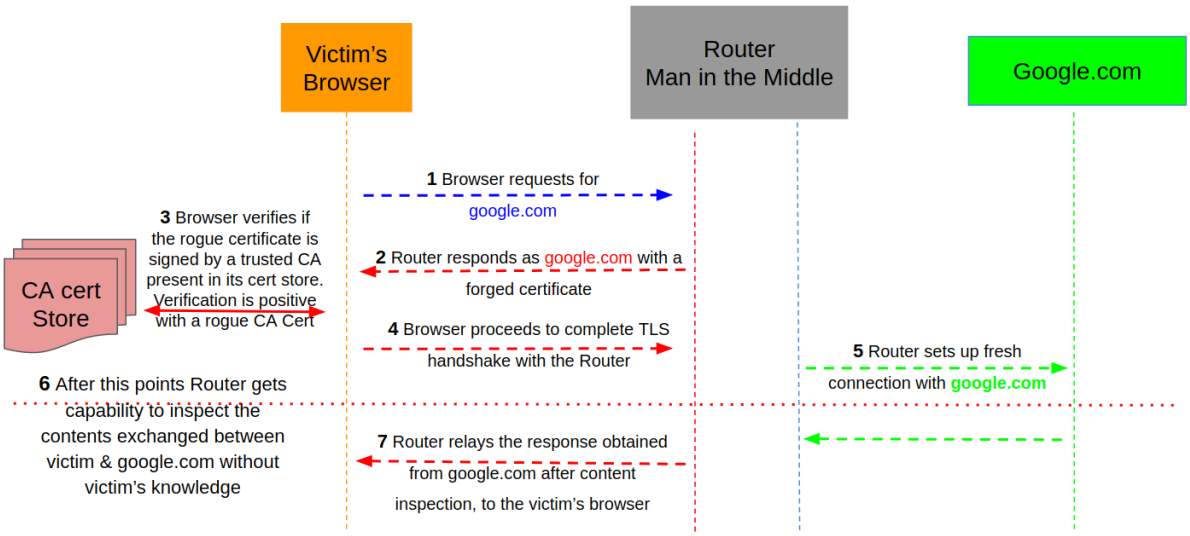
Project Raven involved former NSA employees who performed offensive hacking operations for the UAE. The intermediate CA certificates of Dark Matter, present in the browsers or trust stores, were used for decryption of encrypted communication of chosen targets like human rights activists.

Details

In a typical surveillance project like Raven, the ISPs are in control of a state and the victim's browser already has a CA or intermediate CA certificate in its trust store whose corresponding private keys are with the ISP. Using these private keys an ISP can decrypt any and every communication going through it.

Exploitation

In the illustration below the state controlled ISP is shown as a Router. The attacks is similar to SSL Sniff attack, except that the victim's browser does not warn a user of any possibility of snooping because it trusts the certificate presented by the router.



Explanation of SSLSniff attack with a polluted CA certificate (trust) store

Impact

The impact of spying and snooping on private communications of human right activists & dissidents in UAE has cost many of them life or imprisonment.

Mitigation

It's hard to know the intent of an organization working as an intermediate CA or applying for a root CA hence its difficult to mitigate such incidents. It's also hard to uncover if widely existing CAs or intermediate CAs are involved directly or indirectly, intentionally or unintentionally in snooping on personal communications.

Communities of cryptographers with help of Google & Mozilla foundation are working hard to keep the trust stores on our browsers and machine pristine.

2.3 Buffer overflow

2.3.1 Buffer Overflow in Exim CVE-2019-15846

Introduction

The SMTP Delivery process in all versions including Exim 4.92.1 has a Buffer Overflow. In the default runtime configuration, this is exploitable with crafted Server Name Indication (SNI) data during a TLS negotiation. In other configurations, it is exploitable with a crafted client TLS certificate**.**

Description

Exim server is vulnerable if it accepts TLS connections. This does not depend on the TLS library, so both GnuTLS and OpenSSL are affected.

In order to exploit this vulnerability, the two buffers (input and output), must be aligned to ensure that there are no trailing NULL bytes in between except the NULL byte terminating the input buffer.

More precisely, we have to take care about the store_get that aligns data in a store block on an 8-byte boundary.

Finally, both buffers must belong to the same storeblock to overflow with more than a few bytes. Depending on the shape of the heap when string_unprinting is called, the input buffer can be quite limited in size.

Nevertheless, there is virtually no limit to the amount of data that can be overridden. The read pointer will read the data that have just been written. In order to avoid stopping at the first copy of the original NULL

byte it is possible to prepend it with more backslashes. Thus any address after the output buffer can be reached.

Moreover, it is possible to override with NULL bytes by encoding them with \x00.

Exploitation

The vulnerability is exploitable by sending a SNI ending in a backslash-null sequence during the initial TLS handshake. The Read from Buffer is ignoring the fact that it should stop reading a string when a \0 is hit. Ignoring the char and continuing reading is causing the problem.

This vulnerability is located in string_interpret_escape when it is called by string_unprinting.

pp points a pointer to the initiating "\" in the string; the pointer gets updated to point to the final character.

Returns: the value of the character escape

```
@@ -236,6 +238,7 @@ const uschar *hex_digits=
CUS"0123456789abcdef";

 int ch;

 const uschar *p = *pp;

 ch = *(++p);

 if (isdigit(ch) && ch != '8' && ch != '9')

  {

ch -= '0';
```

As the name suggests string_interpret_escape aims at interpreting escape sequences. For example, \x62 will be converted to b.

string_unprinting uses that function in order to convert an input string into an unescaped output string. The output string is first allocated using the Exim memory allocator :

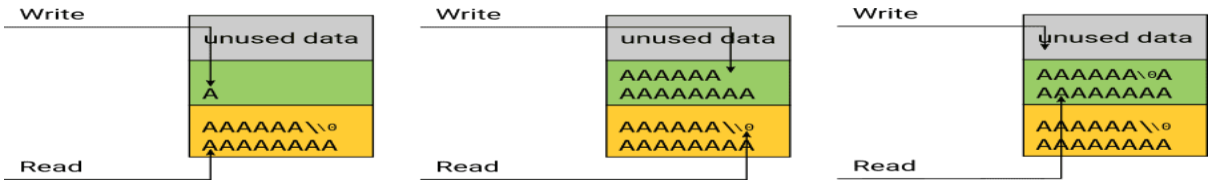
```
len = Ustrlen(s) + 1;

ss = store_get(len);
```

The vulnerability lies in the fact that string_unprinting will read input string until a NULL byte is found. When string_interpret_escape is called, it will move forward the buffer pointer but then string_unprinting will move it again. Thus, it will jump over the char following backslash which results in a copy beyond the limit of the output buffer.

```
while (*p)
 {
 if (*p == '\\')
  {
  *q++ = string_interpret_escape((const uschar **)&p);
                        ignoring Escape and Below pointer ++
  p++;
  }
[...]
}
```

```
@@ -236,6 +238,7 @@ const uschar *hex_digits= CUS"0123456789abcdef";
```




```
int ch;
 const uschar *p = *pp;
 ch = *(++p);
+if (ch == '\0') return **pp;  ----------> Returned after hitting  \0
 if (isdigit(ch) && ch != '8' && ch != '9')
  {
ch -= '0';
```

Impact

According to Exim's vendor advisory report about this issue, any Exim server that accepts TLS connections is vulnerable, and it does not matter what TLS library is being used (GnuTLS or OpenSSL). All versions of Exim's service, up to, but excluding, version 4.92.2, are vulnerable to this issue. Versions 4.92.1 and earlier are susceptible to buffer overflow attacks, which allows an attacker to do virtually anything on your system. This is due to the fact that the bad actor has the same access that you do, or maybe even higher. This is a critical issue with many potential avenues for increasing

Mitigation

Exim has addressed this issue in version 4.92.2 of their MTA. It is crucial to update your Exim service to this version as soon as possible. Additionally, Exim also stated that another mitigation option is to not offer TLS on your server. However, this is not advised as SMTP without security allows anyone who is monitoring the connection to read the information without the need to decrypt it.

References

- https://nvd.nist.gov/vuln/detail/CVE-2019-15846
- https://threatpost.com/critical-exim-flaw-opens-millions-of-servers-to-takeover/148108/
- https://westoahu.hawaii.edu/cyber/vulnerability-research/vulnerabilities-weekly-summaries/cve-2019-15846-exim-mail-transfer-agent-vulnerable-to-buffer-overflow-attack/

2.3.2 TLSv1.3 PSK extension parsing potential buffer overflow

Introduction

A buffer overflow critical vulnerability was found in wolfSSL 4.0.0 embedded SSL/TLS library. The vulnerability is due to improper validation of PSK(PreSharedKey) identity size in the requests. A remote attacker could exploit this vulnerability by sending maliciously crafted requests to a target server. Successful exploitation of this vulnerability could allow a remote attacker to execute arbitrary code on the affected system.

Description

Affected by this vulnerability is the function DoPreSharedKeys of the file tls13.c of the component Hello Packet Handler. The manipulation as part of a Length Field leads to a memory corruption vulnerability.

wolfSSL 4.0.0 has a Buffer Overflow in DoPreSharedKeys in tls13.c when a current identity size is greater than a client identity size. An attacker sends a crafted hello client packet over the network to a TLSv1.3 wolfSSL server. The length fields of the packet: record length, client hello length, total extensions length, PSK extension length, total identity length, and identity length contain their maximum value which is 2^16. The identity data field of the PSK extension of the packet contains the attack data, to be stored in the undefined memory (RAM) of the server. The size of the data is about 65 kB. Possibly the attacker can perform a remote code execution attack.

This vulnerability has been handled as CVE-2019-11873 since 05/10/2019. The attack may be launched remotely. No form of authentication is required for exploitation.

Exploitation

An attacker sends a special crafted hello client packet over the network to a TLSv1.3 wolfSSL server. The length fields of the packet: record length, client hello length, total extensions length, PSK extension length, total identity length and identity length contain their maximum value, which is 65535. The identity data field of the PSK extension of the packet contains the attack data, which the attacker wants to store in the undefined memory (RAM) of the server. The size of the data is about 65 kilobyte. Possibly the attacker can then perform a remote code execution attack.

Impact

Users of wolfSSL v4.0.0 are affected

Mitigation

- Updating to the latest version can fix this issue.
- Please see the references or vendor advisory for more information. Vendors released updates to its Security Gateway R80 / R77 / R75 products

References

- https://www.checkpoint.com/defense/advisories/public/2019/cpai-2019-1559.html
- https://www.wolfssl.com/vulnerability-disclosure-tlsv1-3-psk-extension-parsing-potential-buffer-overflow/
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-11873
- https://www.telekom.com/en/corporate-responsibility/data-protection-data-security/security/details/advisories-504842
- https://www.telekom.com/resource/blob/572524/1c89c1cbaccdf792153063b3a10af10e/dl-190515-remote-buffer-overflow-vulnerability-wolfssl-library-data.pdf
- https://github.com/wolfSSL/wolfssl/pull/2239

2.4 Random Number Generator related flaws

2.4.1 Weakness in Random Number Generation issue in Yubikeys v4.4.2 - 4.4.4

Introduction

The YubiKey FIPS is a FIPS 140-2 certified. The module implements five major functions - Yubico One Time Password (OTP), FIDO Universal 2nd Factor (U2F), PIV-compatible smart card, OpenPGP smart card, and OATH OTP authentication. The firmware version v4.4.2 - v4.4.4 is subjected to reduced randomness of the cryptographic keys it generates. It was found that a few bits of the key remained static, reducing the number of trials to predict the remaining bits of the key.

Description

The buffer holding random values contains contents from the FIPS power-up self-tests, affecting random number generation. These predictable contents are added to the private keys that are being generated.

- The RSA key may be impacted by up to 80 predictable bits out of a minimum of 2048 bits
- ECDSA signatures, the nonce K becomes significantly biased with up to 80 of the 256 bits being static, resulting in weakened signatures
- ECC encryption, 16 bits of the private key becomes known and leaving 240-bits Unknown in secp256r1 and 368 bits unknown in secp384r1.

Exploitation

To successfully exploit this flaw, the attacker has to Intercept the authentication operations and break the rest of the cryptographic key. The exploitation complexity of the flaw can vary based on the number of bits known for the attacker. Upon successful cracking of remaining bits, the attacker can get the private key, leading to exposing sensitive information.

Smart Card	An attacker who gained access to enough signed artifacts to reconstruct the private key used to sign these artifacts
FIDO U2F	An attacker with a small number of weak ECDSA authentication signatures between a FIDO client and a Relying Party could recompute the private key and impersonate a user's U2F Authenticator for that specific Relying Party. The FIDO specification mandates HTTPS, so the attacker must be able to decrypt the TLS communication to perform this attack. An attacker could authenticate as the user and directly sign authentication requests from the Relying Party without the presence of the user's YubiKey FIPS device using the user's private key, username, and password.
OATH One-Time Passwords	Under certain conditions, an attacker can capture the password validation sequence and replay authentication to the YubiKey FIPS device, which allows the attacker to obtain OTP codes.
OpenPGP	Predicting the unknown bits in RSA keys generated on YubiKey FIPS.

Impact

The company recalled all the vulnerable version keys and replaced them with the updated version of the keys for their customers.

YubiKey FIPS Series with firmware 4.4.2 and 4.4.4

- YubiKey FIPS
- YubiKey Nano FIPS
- YubiKey C FIPS
- YubiKey C Nano FIPS

Mitigation

- This issue will occur only for a few operations. Once the buffer is exhausted, the normal random generation algorithm does not have any predictable contents in the keys.
- This issue is fixed in v4.4.5 firmwide. Updating the firmwide to the version with the vulnerability fix helps in mitigating the flaw.

References

- https://www.yubico.com/support/security-advisories/ysa-2019-02/
- https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf

2.4.2 The Linux kernel's getrandom() function vulnerability

Introduction

getrandom() is the method used in Linux to generate random values that can be used for the application that requires random values. This method was introduced to address the issues caused by urandom with:

- The needs to access filesystem paths, which can fail, e.g. under a chroot
- The needs to open a file descriptor, which can fail under file descriptor exhaustion attacks
- The possibility to get not-so-random data from /dev/urandom, due to an incompletely initialized kernel entropy pool

Since the developers are not aware of the internal working of the getrandom() method, the application that uses this method can wait indefinitely if this method is not handled properly.

Vulnerability Description

The property of the getrandom() method causes the Linux boot up to hang indefinitely. The getrandom() was made to block until a proper amount of entropy has been accumulated. This made the system call have no guaranteed upper-bound for its waiting time. If any of the bootup processes use this in init or part of code the execution will be waiting till the entropy is met. To overcome this issue GRND_NONBLOCK has to be used, and a fallback to some other interface like /dev/urandom is required, thus making the net result no better than just using /dev/urandom. This Vulnerability can introduce the following bugs.

- Failure in accessing filesystem paths under chroot
- File descriptor exhaustion attacks can be performed, where the attacker consumes all available file descriptors
- Forcing the use of the fallback code where /dev/urandom is not available
- The randomness of the data generated is very less thus making it weak and predictable.

Exploitation

The attacker can create a program that uses the random method and make it execute during the bootup process. This program hangs the system, thus affecting the availability of the system to the user.

If the user is using the vulnerable getrandom() for the creation of private keys and if the attackers are somehow able to capture the keys. The attacker who is having a sufficient number of private keys can perform cryptanalysis, thus leading to the leaking of secret messages.

The attacker can use CVE-2018-1108 exploitation on the unfixed version

Impact

Kernel drivers before version 4.17-rc1 are vulnerable to a weakness in the Linux kernel's implementation of random seed data. Programs, early in the boot sequence, could use the data allocated for the seed before it was sufficiently generated.

- Redhat Enterprise Mrg 2
- Redhat Enterprise Linux 7
- Linux kernel Up to 4.16

Mitigation

- Don't trust user-space on calling getrandom(2).
- Never block, by default, and just return data from the urandom source if entropy is not yet available.
- Issue a big WARN_ON when any process gets stuck on getrandom(2) for more than CONFIG_GETRANDOM_WAIT_THRESHOLD_SEC seconds.
- Introduce the new getrandom2(2) system call, with clear semantics that can guide user-space in doing the right thing.
- Check for repeated, weak, and factorable keys, if it is found predictable generate a new set of keys.
- The key must not be generated during the boot as there is a possibility that it generates predictable keys.
- The OS should provide an interface to indicate how much entropy it has mixed into its random number so that applications can gauge whether the state is sufficiently secure for their needs.

References

- https://lore.kernel.org/linux-ext4/20190915081747.GA1058@darwi-home-pc/
- https://lkml.org/lkml/2019/9/18/1025
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1108

2.4.3 Cache side-channel attacks on the widely used random number generator CTR_DRBG

Introduction

Recent times we've seen plenty of attacks against pseudo-random number generators which form a critical part of any crypto system as many have demonstrated when an RNG is compromised it can result in devastating attacks against the system of which it is a part.

Plenty of realworld, random number generator disasters:

- Dual_EC Backdoor
- Juniper Dual_EC Incident
- DUHK Attack on ANSI X9.31

Many of these designs are standardised in a set of publications under NIST SP 800-90 series list approved designs approved by NIST the US National Institute of Standards and Technology:

- Dual_EC (Deprecated following internal investigation)
- HMAC_DRBG
- HASH_DRBG
- CTR_DRBG

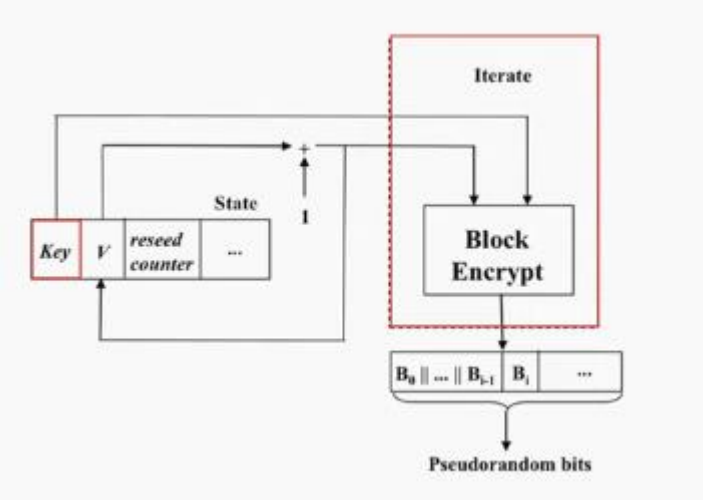
Description

The CTR_DRBG, a design standardized in NIST Special Publication(SP) 800 90A (and the Federal Information Standards program to which it belongs) omit side channel attacks from their threat model. As a result, manufacturers are compliant even if they use a non-side-channel-secure AES implementation.

Counter DRBGs design includes a state which is composed of the key K and a counter V. The broad mechanic of the design revolves around incrementing the counter which the algorithm then encrypts with the blog cipher taking the ciphertext as the output of the RNG. The standard includes an option for the user or the implementer of the RNG to provide additional entropy which is encrypted along with the counter in order to provide protection against state compromise.

Key rotation flaw:

- **Problem 1:** Key is not rotated until after encryptions are done. Allow large no.of encryptions on same keys. If the attacker compromises the key at any point during this phase, might be able to recover the RNG internal state.
- **Problem 2:** Additional entropy is optional and implementer chosen. Implementer may ignore it altogether.



Exploitation

AES block cipher used in most CTR_DRBG implementations is well known to possess side channel vulnerability if implemented in software. "T-table" AES has been a canonical target for cache attacks for 15 years. Most crypto libraries now use AES-NI and hardware based AES by default. Most implementations use hardware AES when the user requests encryption of string.

The side-channel attack against CTR_DRBG is performed to recover secret keys from TLS sessions. Authors found a protocol flaw within TLS that forces a client to generate enough PRG output to run the attack, and allows us to calculate the private key used by the client to authenticate to the server. Empirically, cache attacks require around 2000 bytes of AES output to recover the key.

Steps used:

- Used FLUSH+RELOAD attack on T-Table AES
- Brute forced additional entropy
- From ECDSA signature nonce, computed victims ECDSA private key

Our attacker can now masquerade as the client by generating signatures. Complexity of attack depends on implementers choice for additional entropy.

Impact

Vulnerable AES CTR_DBG implementations

- NetBSD kernel systemwide PRG
- OpenSSL 1.0.2 FIPS module
- FortiOS v5
- mbedTLS-SGX
- nist_rng library

Mitigation

Random number generators can be side-channeled too. Take below measures:

1. Don’t use T-table AES anywhere
2. Add additional entropy
3. Prediction resistance

FIPS 140-3, updates threat model (effective Sep 2019) to incorporate side channels. When choosing what RNG to use, if you are going to choose from NIST SP 800-90A, use Hash_DRBG (HMAC_DRBG also has problems)

References

- Research Post: Cache Attacks on CTR_DRBG
  https://security.cohney.info/blackswans/
- Pseudorandom Black Swans: Cache Attacks on CTR_DRBG
  https://www.youtube.com/watch?v=MZVJyVnU2RE
  https://www.youtube.com/watch?v=T9e3a8XiIe8

2.5 Padding Oracle

2.5.1 Zombie POODLE, GOLDENDOODLE & K10065173: TMM TLS virtual server vulnerability CVE-2019-6593

Introduction

A Zombie POODLE and GOLDENDOODLE is where the server behaves differently while decrypting the improper padded data received from the user/attacker. This attack is performed on the CBC encryption mode that uses the padding technique like PKCS#7. There is a possibility that the server can reveal valuable information as a response to the request. An attacker can relocate a ciphertext block to replace a block of padding that can decrypt the last byte of the targeted block if the server confirms proper padding.

Description

Zombie POODLE is also known as POODLE BITES or POODLE 2.0 attack. In this exploit, server stacks behave differently when receiving TLS records with valid MAC and invalid padding. Zombie POODLE typically implies that the implementation did validate padding bytes but inadvertently leaked the result.

GOLDENDOODLE attack is similar to the Zombie POODLE exploit. Here the attacker adds a guess G at the MAC part of the request since it is not validated correctly. This guess can influence the decryption process and can be used to find the plaintext. This method takes fewer requests than POODLE due to MAC validation. In this attack, the attacker has more control to manipulate the ciphertext without breaking the oracle response. GOLDENDOODLE oracles allow the attacker to confirm guessed values for plaintext bytes rather than waiting for random occurrences.

These vulnerabilities are applicable only if the server uses TLS 1.2 or TLS 1.1 or TLS 1.0 with CBC cipher modes. The attacker can also get the details of padding validity, MAC validity, specific bit value, plaintext length.

Exploitation

The attacker must trigger a victim’s requests to an authenticated resource while actively intercepting and modifying the resulting SSL records. In other words, the victim’s web browser must have authenticated session cookies for a service using a vulnerable TLS stack. The attacker must be a MITM to observe the change of the responses. If the above condition is met, then this can be exploited.

Step1 : Attacker must be in the MITM position before the TLS handshake.

Step2 : Attacker must execute JS <Java Script> from the victim's browser.

Step3 : Padding block replaced with some random bytes, cookies, and sent to the server. In the case of Goldendoodle, the attacker must add guess bytes after/at the MAC position.

Step4 : The server decrypts the data; the padding location will have random values.

Step5 : If TLS alerts are sent back as a response, padding is valid. Based on this response, the attacker can brute force and decrypt the plaintext.

Goldendoodle example equation: C6' = C6[0...14]|(G XOR C3[15]) where G is the guess for P4[15], C6 - ciphertext 6, P4 plaintext 4. If the guess is correct, we will get the result as 0. It decrypts 1 byte per requests for 255 request it takes average 255 requests/bytes.

Impact

Over 5800 domains are readily exploitable, and around 1.6% of the TLS enabled Alexa top 100K are vulnerable to this attack. Exploiting this vulnerability leads to plaintext recovery of encrypted data and may also reveal session authentication cookies.

Mitigation

- MACing the data after encryption and validating the MAC after decryption can mitigate this issue.
- Use the latest TLS version.
- Review your CBC code and make sure it does not expose a padding oracle.
- Make sure TLS implementations do not reveal any information about padding validity, alerts, connection states, or even timing behavior.
- To check if the servers are vulnerable to Zombie Poodle attack and Goldendoodle tools like Tripwire/padcheck and RUB-NDS/TLS-Scanner from the GitHub can be used.

References

https://www.tripwire.com/state-of-security/vulnerability-management/zombie-poodle-goldendoodle/

2.5.2 New variant of Bleichenbacher attacks against RSA in TLS using Cache side channels

Introduction

Bleichenbacher's attack is also called the Million Messages attack. It is an adaptive chosen-ciphertext attack on the RSA PKCS#1 v1.5 encryption padding scheme. The presence of a side-channel at the TLS server which allows distinguishing PKCS#1 compliant from non-compliant ciphertexts. An attacker with access to the side-channel can record the target's TLS handshake and extracts the RSA-PKCS#1 encrypted ClientKeyExchange message c that contains encrypted PremasterSecret. The PremasterSecret is used to derive all TLS session keys. The decryption of PremasterSecretdecr leads to the decryption of the TLS traffic. The attacker creates new ciphertexts from c, these are sent to the TLS server as part of a new handshake, and the server's responses are observed. With each successful query, the attacker can reduce

the interval in which the original plaintext is located. The attacker repeats these steps until the interval only contains one integer and thus decrypting the ciphertext c.

Description

BEAST mode is used with Man in the Browser attack by using Cache-like attacks to perform a downgrade attack against any TLS connection to a vulnerable server.

Other possible attacks on side channels:

1. The implementation bug in the Java Secure Socket Extension (JSSE) – Java's built-in SSL/TLS implementation. In JSSE, a different error message can be triggered if the two most significant bytes are PKCS#1 compliant, but the PreMasterSecret shows up to be of invalid length. They can be exploited and decrypt a PreMasterSecret with a few thousand queries.
2. Timing difference in OpenSSL implementation during PKCS#1 processing causes a Side-channel attack. This side channel's source is hard to determine: The timing difference is caused by the unequal treatment of random number generation may be the cause for this side channel.
3. Java's Exception handling and error processing can be a time-consuming task. Whenever the resulting plaintext is not PKCS#1 compliant, an Exception is raised by JSSE forcing random PreMasterSecret generation. The resulting timing difference is significantly higher and can be measured over a LAN.

4. F5 BIG-IP and IBM Datapower products, which rely on the Cavium NITROX SSL accelerator chip. It allows you to distinguish invalid messages from messages starting with 0x?? 02 (where 0x?? represents an arbitrary byte) since the original Bleichenbacher algorithm does not correctly handle this case.

Exploitation

- Sniff the TLS handshake between the client and the server and the first message.
- Use the Bleichenbacher algorithm to decrypt the premaster secret.
- Decrypt the first message.
- Get the cookie.

The attacker requires three capabilities:

1. Side-channel capability: the attacker must execute code on the victim machine to mount a microarchitectural side-channel attack.
2. Privileged Network Position Capability: The attacker must be in the man-in-the-middle position so that padding oracle attacks with the private key can be exploited. The attacker downgrades the communication to TLS 1.2 RSA key exchange and uses the BEAST method.
3. Decryption Capability: The attacker needs an ability to start the decryption of ciphertext chosen by him on the target system. They utilized Bleichenbacher and Manger's Attack attack.

A concrete scenario of the attack:

- A physical device is shared between a TLS server and the attacker's virtual machine. This requires a determined adversary.
- This can be achieved by controlling a node between the server and the client.

Impact

More than 6% of the internet are vulnerable to the attack. The vulnerability was exploited in the following SSL implementation.

- OpenSSL
- OpenSSl API
- Amazon s2n
- MbedTLS
- Apple corTLS
- Mozilla NSS
- WolSSL
- GnuTLS

Mitigation

- Use the latest TLS version.
- Choose better algorithms like Ephemeral ECDH.
- The backward compatibility is the main issue of this attack. TLS 1.3 security doesn't help you if your server can be downgraded to an earlier version.
- Deprecate the RSA key-exchange.
- Don't use the same key for signing and RSA-key exchange.
- Constant-Time Code and Safe API as in BearSLL and BoringSSL API.
- Using Large RSA Keys.
- Reduce the TLS Handshake Timeouts to make MitM attack harder.
- Use BearSSL or BoringSSL

References

- https://owasp.org/www-pdf-archive/2014-07-bleichenbacher-ssl.pdf
- https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-meyer.pdf
- https://eprint.iacr.org/2018/1173.pdf

2.6 Other Categories

2.6.1 Serial number mistake

Introduction

Membership in the root certificate program is the way in which Mozilla decides which certificate authorities (CAs) get to have their root certificates trusted in Firefox. Mozilla's list of trusted root certificates is also used in many other products, including the Linux operating system. During a discussion about the DarkMatter CA on a Mozilla mailing list, it was found that their 64-bit serial numbers weren't actually 64 bits, and it opened a can of worms. It turns out that the serial number was effectively 63 bits, which is a violation of the CA/B Forum Baseline Requirements that state it must contain 64 bits of output from a secure random number generator (CSPRNG). As a result of this finding, 2,000,000 certificates or more may need to be replaced by Google, Apple, GoDaddy and various others. It's quite likely that the full scope of this problem hasn't been determined yet.

Description

During an analysis of certificates issued by DarkMatter, it was found that they all had a length of exactly 64 bits – not more, not less. If there's a rule that requires 64 bits of CSPRNG output, and the serial number is always 64 bits, at first glance this seems fine.

But, there's a problem, and it's in RFC 5280; it specifies the following: The serial number MUST be a **positive integer** assigned by the CA to each certificate. It MUST be unique for each certificate issued by a given CA (i.e., the issuer name and serial number identify a unique certificate). CAs MUST force the serialNumber to be a non-negative integer.

Requiring a positive integer means that the high bit can't be set – if it is set, it can't be used directly as a certificate serial number. As such, if the high bit is set, there are two possible options:

1. Pad the serial with an additional byte, so that the full 64 bits of output is used.
2. Discard the value, and try again until you get a value without the high bit set. This means that the size is always 64 bits, and the high bit is always 0 – giving you 63 effective bits of output.

A popular software package for CAs, EJBCA had a default of using 64-bit serial numbers, and used the second strategy for dealing with CSPRNG output with the high bit set. This means that instead of using the full 64-bit output, it effectively reduced it to 63 bits – cutting the number of possible values in half. When we are talking about numbers this large, it's easy to think that 1 bit wouldn't make much difference, but the difference between 2^64 and 2^63 is substantial – to be specific, 2^63 is off by over 9 quintillion or more specifically 9,223,372,036,854,775,808.

The strategy of calling the CSPRNG until you get a value that has the high bit unset violates the intention of the rule imposed by the Baseline Requirements, meaning that all certificates issued using this method were mis-issued. This is a big deal, at least for a few CAs and their customers.

Now, the simple solution to this is to just increase the length of the serial beyond 64 bits; for CAs that used 72 or more bits of CSPRNG output, this is a non-issue, as even if they coerce the high bit, they are still well above the 64-bit minimum. This is a clear case of following a standard as close to the minimum as possible, which left no margin for error. As the holders of those 2+ million certificates are learning, they cut it too close.

The Baseline Requirements are the minimum rules that all CAs must follow; these rules are voted on by a group of browser makers and CAs, and often debated in detail. Effective September 30, 2016, CAs SHALL generate non-sequential Certificate serial numbers greater than zero (0) **containing at least 64 bits of output from a CSPRNG**. In this case, the fact that 1 bit would be lost in a purely random serial was pointed out by Ryan Sleevi of Google and Ben Wilson of DigiCert. With a deeper reading, it's clear that a 64-bit serial, the smallest permitted, is quite likely to be a violation of the Baseline Requirements. While you can't look at a single certificate to determine this, looking at a larger group will reveal if the certificate serial numbers are consistently 64 bits, in which case, there could be a problem.

Failing to comply with the Baseline Requirements will complicate audits, and could put a CA at risk of being removed from root stores.

Exploitation

Entropy in the serial number is required as a way to prevent hash collisions from being used to forge certificates; this requires an ability to predict or control certificate contents and the use of a flawed hashing algorithm, adding a random value makes this more difficult.

This type of issue has been exploited with MD5, and could someday be exploited with SHA1; there's no known flaws in the SHA2 family (used in all current end-entity certificates) that would allow such an attack.

In addition, while due to this issue, the level of protection is reduced by half, 2^63 is still a large number and provides a substantial amount of safety margin.

Impact

It's also not clear how many other CAs may be impacted by this issue; while a few have come forward. For Google and Apple, both in the process of replacing their mis-issued certificates, they were only issued to their own organizations – reducing the impact. On the other hand GoDaddy, which has mis-issued more than 1.8 million certificates, faced a much larger problem as these were certificates issued to customers. Customers that are likely managing their certificates manually, and required substantially longer to complete the process.

Mitigation

When a certificate is issued that doesn't meet the Baseline Requirements, the issuing CA is required to take quick action. Baseline Requirements (4.9.1.1) guidance state "The CA SHOULD revoke a certificate within 24 hours and MUST revoke a Certificate within 5 days if one or more of the following occurs. The CA is made aware that the Certificate was not issued in accordance with these Requirements or the CA's Certificate Policy or Certification Practice Statement"

This makes it clear that the CA has to revoke any certificate that wasn't properly issued within 5 days. As a result, CAs are under pressure to address this issue as quickly as possible – replacing and revoking certificates with minimal delay to avoid missing this deadline.

Google was able to revoke approximately 95% of their mis-issued certificates within the 5 days. Apple announced that they wouldn't be able to complete the process within 5 days. GoDaddy stated that they would need 30 days to complete the process. The same reason was cited by all three: minimizing impact. Without robust automation, changing certificates can be complex and time-consuming, leaving the CA to choose between complying with requirements or impacting their customers.

References

1. DarkMatterConcerns https://groups.google.com/g/mozilla.dev.security.policy/c/nnLVNfqgz7g
2. Baseline Requirement Documents (SSL/TLS Server Certificates) https://cabforum.org/baseline-requirements-documents/
3. Cyber-Mercenary Groups Shouldn't be Trusted in Your Browser or Anywhere Else https://www.eff.org/deeplinks/2019/02/cyber-mercenary-groups-shouldnt-be-trusted-your-browser-or-anywhere-else
4. RFC 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. https://tools.ietf.org/html/rfc5280#section-4.1.2.2
5. DarkMatterConcerns: https://groups.google.com/u/1/g/mozilla.dev.security.policy/c/nnLVNfqgz7g/m/VAdQotoiBQAJ

2.6.2 Outage due to expired certs

Introduction

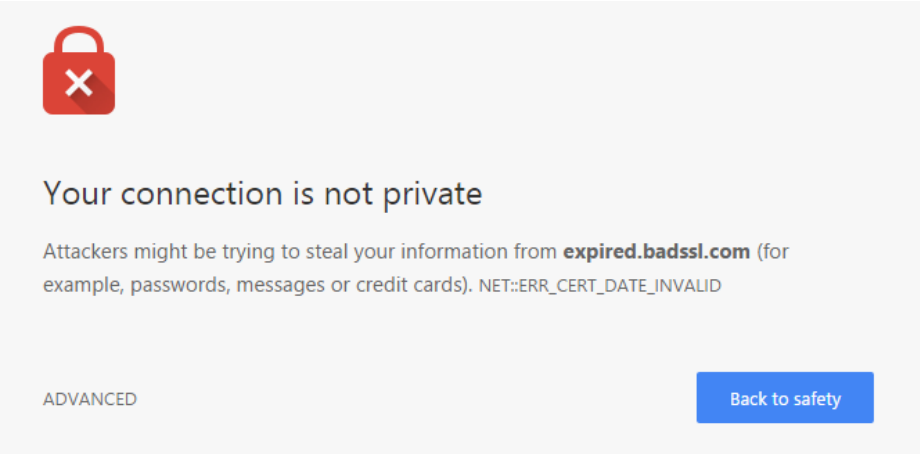
On 6th of December 2018, millions of O2 mobile subscribers in the UK were knocked off the cellular service due to the expiration of a digital certificate at a connecting node with Ericsson's network.

According to Ericsson's update, initial root cause analysis indicated that the main issue was an expired certificate in the software versions installed with these customers. A complete and comprehensive root cause was due by early 2019.

Description

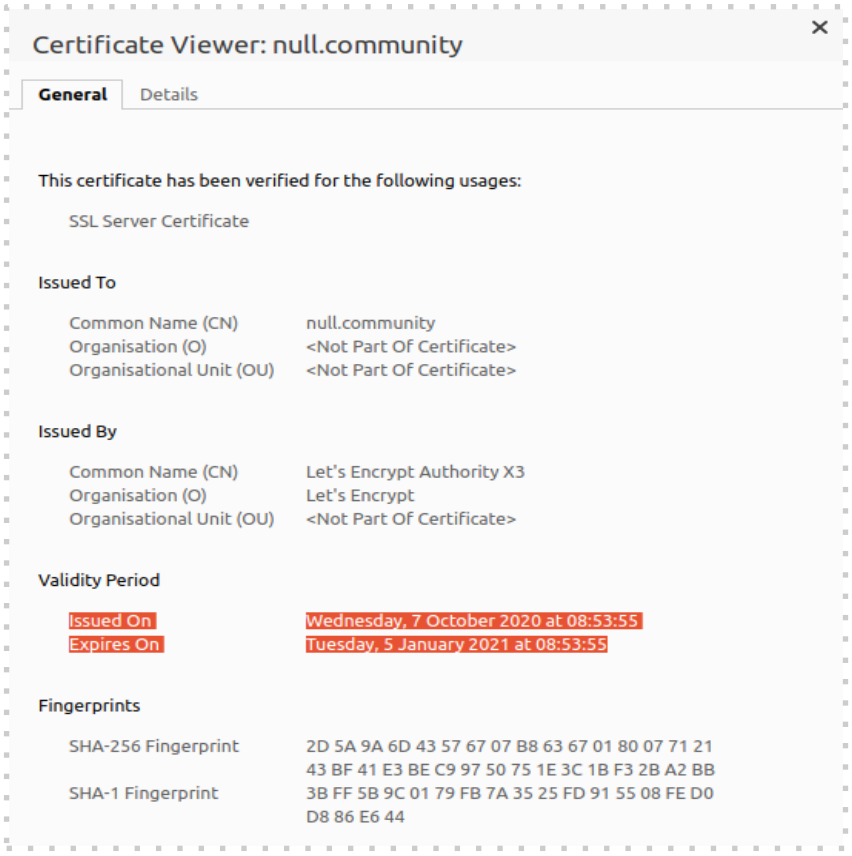
X509 certificates are also used for mutual authentication and securing the machine-to-machine communication. This forms the backbone of trust on which today's internet operates.

X509 certificates are issued with a validity period marked by "*issued on*" or "*not valid before*" time stamp and "*not valid after*" or "*expires on*" timestamp as you can see in the below capture of ***null.co.in*** certificate.



Browser warning about expired peer certificate

If any node in the communication presents an expired certificate during the SSL/TLS handshake the other node will reject the connection resulting in outage and lack of connectivity.



X509 certificate of https://null.co.in

Exploitation

The Public Key Infrastructure (PKI) is designed to work on trusted certificates. A definite validity span of a certificate is at the root of this trust based system. Using expired certificates results in denial of service affecting revenue and reputation of an organisation. This is a kind of attack which cannot be exploited by external adversaries but is resulted due to lack of standardization of operating procedures and adherence to it.

Impact

The 40 minutes outage caused inconvenience to 7 millions of O2 subscribers. Ericson assured to compensate the O2 users with extra toups costing in millions.

In the past XBOX live and Azure service have suffered downtime due to expired certificates.

Mitigation

All certificates provisioned to all the nodes (server or machines) must be monitored & periodically refreshed. Rotating the keys and refreshing the keys must be an important part of the standard operating procedure.

References

- A complete study of P.K.I. (PKI's Known Incidents)
- https://www.feistyduck.com/bulletproof-tls-newsletter/
- https://bugzilla.mozilla.org/show_bug.cgi?id=1544933
- https://www.theverge.com/2018/12/7/18130323/ericsson-software-certificate-o2-softbank-uk-japan-smartphone-4g-network-outage
- https://www.bbc.com/news/business-46499366
- https://www.reuters.com/investigates/special-report/usa-spying-raven/
- https://www.ericsson.com/en/press-releases/2018/12/update-on-software-issue-impacting-certain-customers

2.6.3 Golang TLS flaw

Introduction

The code written in Go Lang and using one-way or mutual TLS authentication, is vulnerable to CPU denial of service attack.

Description

Package crypto/X509 of Golang:

1. Parses and validates X.509-encoded keys and certificates
2. It is supposed to handle certificate chains provided by an attacker with reasonable resource use.

Problem:

The crypto/x509 package does not limit the amount of work performed for each chain verification, which might allow attackers to craft pathological inputs leading to a CPU denial of service. Go TLS servers accepting client certificates and TLS clients verifying certificates are affected.

1. As the package parses the client certificate and checks the validity of the certificate, it needs to pull all CA (Root certificates) for comparison.
2. Also, it needs to parse the intermediate certificates for the chain validation.
3. In this chain validation, just for 1 single TLS connection, 100% CPU resources were utilized because of inefficient algorithms and handling in the chain validation procedure in Golang itself.
4. Thus, If an attacker crafted certificates based on this vulnerability(example-> if attacker crafted a client certificate as client1->client2(intermediate)->client3(intermediate)->root then CPU resource utilization would be 100% leading to DOS attack.

Exploitation

GoLang inherently has functions used to verify the chain validation of certificates in order to establish TLS connection. However, as these functions are not implemented correctly, while doing a chain validation, in case of intermediate certificates, it consumes all CPU performance and thus are vulnerable to DoS attack.

The main CPU DoS is triggered by **buildChains**() and **findVerifiedParent**() functions in the unexpected conditions where all intermediate CA certificates share the same name and have a nil AuthKeyId value. The **findVerifiedParent**() function returns all certificates matching that name, which is the entire pool, and then checks signatures against all the certificates. Once that is done, the **buildchains**() function is again called for the found parent recursively until it reaches the root CA, each time verifying against the entire intermediate CA pool ,and hence consuming all available CPU for only one TLS connection.

- An attacker can construct a certificate chain that makes the client certificate verification consume all the CPU resources and thus making the host less responsive.

Henceforth, whoever is using the version of GoLang and using the functionality , are vulnerable to DoS attack.

Impact

- This has been implemented with only one connection.
- In accordance with Go scheduler rules, only two CPU cores were affected and used at 100%, creating a new connection and forcing the scheduler to allocate more resources to process the signature check, which in turn can lead to an unresponsive service or host.

Mitigation

To get the fixes, upgrade immediately to Go v1.10.6 or later, or v1.11.3 or later.

References

https://apisecurity.io/mutual-tls-authentication-vulnerability-in-go-cve-2018-16875/

2.6.4 Go Salsa20 Package

Salsa20 is a stream cipher that works on data blocks of size of 64 bytes. The encryption algorithm works on a 64-byte block of data. The input to the **expansion function** is the secret key (which can have either 32 or 16 bytes) and an 8-byte long nonce concatenated with an additional block number, which values change from 0 to 264-1 (it is also stored as 8 bytes). Every call to the expansion function increases the block number by one.

The core of Salsa20 encryption algorithm is a **hash function** that receives the 64-byte long input data from the Salsa20 expansion function, mixes it, and eventually returns the 64-byte long output. The output from the Salsa20 expansion function is XOR to the 64-byte block of data. The result is a 64-byte block of ciphertext.

For decryption, the salsa20 expansion function's output should be XORed to the ciphertext's 64-byte block. The result is a 64-byte block of plaintext.

Introduction

Package salsa20 implements the Salsa20 stream cipher as specified in https://cr.yp.to/snuffle/spec.pdf.

This package also implements XSalsa20: a version of Salsa20 with a 24-byte nonce as specified in https://cr.yp.to/snuffle/xsalsa-20081128.pdf. They are merely passing a 24-byte slice as the nonce triggers XSalsa20.

The vulnerability was classified as critical. The manipulation of a *Keystream* led to weak encryption. Using CWE to declare the problem leads to CWE-330. The potential impact of an exploit of this vulnerability is considered to have a high impact on confidentiality, with no impact on integrity and availability.

Description

An issue was discovered in supplementary Go cryptography libraries. A flaw was found in the amd64 implementation of golang.org/x/crypto/salsa20 and golang.org/x/crypto/salsa20/salsa. Suppose more than 256 GiB of the keystream is generated, or the counter otherwise grows greater than 32 bits. In that case, the amd64 implementation will first generate incorrect output and then cycle back to the previously generated keystream. Since there is an issue with the randomness of the

keystream, the repetition of keystream bytes can lead to loss of confidentiality in encryption applications or predictability in CSPRNG applications. This issue was raised due to a logic flaw in a low-level assembly code implementation.

Architectures other than amd64 and uses that generate less than 256 GiB of keystream for a single salsa20.XORKeyStream invocation is unaffected.

Exploitation

- The exploitation is known to be complicated as it requires a large number of encrypted messages to perform cryptanalysis.
- Once the keystream is exhausted, the attacker can predict the keys which will be used in the next message. With the predicted key, encrypted message, and known algorithm, the attacker can decrypt the message and read the plaintext.
- CVE-2019-11840, https://github.com/mmcloughlin/cryptofuzz can be modified to exploit the vulnerability.
- The attack may be launched remotely, and no authentication is required for exploitation.

Impact

All the software/Libraries that use golang-googlecode-go-crypto and that have AMD64 version 2019-03-20 are vulnerable to this attack.

The issue might affect the uses of golang.org/x/crypto/nacl with humongous messages. The vulnerable code is derived from the amd64-xmm5 and amd64-xmm6 implementations that are distributed with SUPERCOP, NaCl, and at https://cr.yp.to/snuffle.html.

Mitigation

- https://github.com/mmcloughlin/cryptofuzz can be used to find if your package version is running on a vulnerable version.
- The vulnerability is fixed. Updating and upgrading the package to the latest version can mitigate this issue.
- Low-level assembly code implementation must be carefully programmed because it may lead to a contentious disclosure process.

References

- https://en.wikipedia.org/wiki/Salsa20
- https://cr.yp.to/snuffle/spec.pdf
- https://search.gocenter.io/golang.org/x/crypto?tab=godocs
- https://github.com/mmcloughlin/cryptofuzz

2.6.5 Invalid curve use

Introduction

AMD Secure Encrypted Virtualization (SEV) is a hardware memory encryption feature. SEV protects guest virtual machines from the hypervisor, provides confidentiality guarantees at runtime and remote attestation at launch time. SEV key management code runs inside the Platform Security Processor (PSP)

Description

Secure Encrypted Virtualization (SEV) on Advanced Micro Devices (AMD) Platform Security Processor (PSP; aka AMD Secure Processor or AMD-SP) 0.17 build 11 and earlier has an insecure cryptographic implementation.

The SEV elliptic-curve (ECC) implementation was found to be vulnerable to an invalid curve attack. This vulnerability was discovered and reported to AMD by Cfir Cohen of the Google Cloud security team.

The weakness was published 06/25/2019 as confirmed advisory (Website). The advisory is available at amd.com. This vulnerability has been traded as CVE-2019-9836 since 03/15/2019.

Exploitation

At launch-start command, an attacker can send small order ECC points not on the official NIST curves, and force the SEV firmware to multiply a small order point by the firmware's private DH scalar.

By collecting enough modular residues, an attacker can recover the complete PDH private key. With the PDH, an attacker can recover the session key and the VM's launch secret. This breaks the confidentiality guarantees offered by SEV.

ECC point multiplication relies on a point addition primitive. There are different implementations for ECC point addition. A common one is based on the short Weierstrass ECC form Note that the curve's "b" equation parameter is never used. An invalid curve attack is where the ECDH point multiplication is done on a different curve - different (a,b) parameters. This becomes possible in the short Weierstrass point addition function since the "b" parameter is not used. On this curve, the point has a

small prime order. By trying all possible values for the small order point, an attacker can recover the private scalar bits (modulo the order). The modular residues are assembled offline using the Chinese Remainder Theorem, leading to a full key recovery.

The exploitability is told to be difficult. It is possible to launch the attack remotely. The exploitation doesn't require any form of authentication. The technical details are unknown and an exploit is not available

Impact

Through ongoing collaboration with industry researchers AMD became aware that, If using the user-selectable AMD secure encryption feature on a virtual machine running the Linux operating system, an encryption key could be compromised by manipulating the encryption technology's behavior.

AMD EPYC server platforms (codename "Naples") running SEV firmware version 0.17 build 11 and below are affected.

Mitigation

Upgrading eliminates this vulnerability. The upgrade is hosted for download at developer.amd.com. A possible mitigation has been published immediately after the disclosure of the vulnerability. The advisory contains the following remark:

AMD released firmware-based cryptography updates to ecosystem partners and on the AMD website to remediate this risk.

References

- AMD-SEV: Platform DH key recovery via invalid curve attack
- https://seclists.org/fulldisclosure/2019/Jun/46
- AMD Secure Encrypted Virtualization (SEV) https://developer.amd.com/wp-content/resources/55766.PDF
- https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9836

2.6.6 Snooping by AVAST

Introduction

Eric Lawrance in his blog titled Spying on HTTPS mentioned how popular antivirus are using browser extensions to inject code to dump the encryption keys on disk to inspect the TLS communication for possible threats.

According to the blog Antivirus developers have two common techniques to scan content running in the browser:

1. Network interception - you can find details of this method in my report "*Secure Usage & Management of X509 Certificates*".
2. Code injection - this method uses a browser extension to dump the session keys maintained in browser's memory to disk via named pipes.

Details

Where are my keys

As a result of TLS handshake, browser & the remote server negotiate a session key to encrypt the subsequent communication.

Browser holds all such session keys in the browser's process memory. Browsers (Firefox & Chrome) checks for a particular environment variable SSLKEYLOGFILE on the system. When the browser finds this variable set, it writes the session keys in a log file pointed by this environment variable.

Exploitation

Adversaries can set the environment variable SSLKEYLOGFILE and force a system to dump the session keys and parameters used in negotiation of it.

On linux machine:

You can set SSLKEYLOGFILE environment variable from a terminal as follows

$ export SSLKEYLOGFILE=~/myLab/sslkeylog.txt

After this fire and instance of Firefox on the same terminal session

$ firefox &

Open a few sites in the firefox to generate some TLS traffic.

Next you can see the session keys and parameters used to negotiate this keys written in the file with simple cat command

$ cat sslkeylog.txt

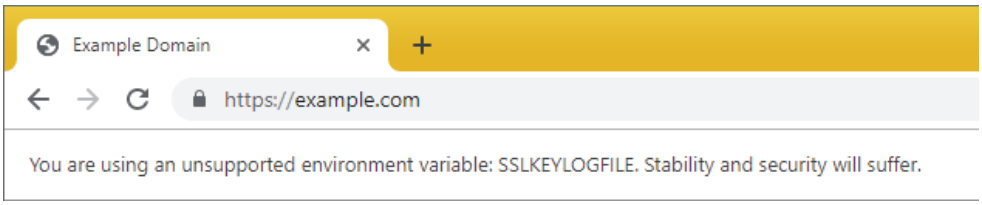
```
evader@pensive:~/myLab$ cat sslkeylog.txt
# SSL/TLS secrets log file, generated by NSS
CLIENT_RANDOM caea19769072f00c582fb7c8fe31d2b1aefe5e19ef9cb918583e1904037bedec 2e0f02373b960d99969f0164e89f1bd5413ad68fd75add022dda1de23b15dc9
2f8cd33ab8ae71b7a5409d31bc0145299
CLIENT_HANDSHAKE_TRAFFIC_SECRET c3afe29f8b8ea218d097587d22ccffaade8ef3839348127904fccc8e2b4b4b1d c1580cb5c3237f9847f32da5975c5f0aa7a749b60da14
6b5b5b0f9ea2b49651d
SERVER_HANDSHAKE_TRAFFIC_SECRET c3afe29f8b8ea218d097587d22ccffaade8ef3839348127904fccc8e2b4b4b1d 08f638cfffc98a15b2a7923b392c2406a9a9ed6d44072
873448e960b6f4ca963
CLIENT_TRAFFIC_SECRET_0 c3afe29f8b8ea218d097587d22ccffaade8ef3839348127904fccc8e2b4b4b1d 2b8da1c58423512f5b6d987dcf26bbd45705ae934cafc7e045187
10116a46603
SERVER_TRAFFIC_SECRET_0 c3afe29f8b8ea218d097587d22ccffaade8ef3839348127904fccc8e2b4b4b1d 25d8f2406bdae1952da78d8626f5e5c6c59632494781d1cbaa8db
f3c1db0e0c3
EXPORTER_SECRET c3afe29f8b8ea218d097587d22ccffaade8ef3839348127904fccc8e2b4b4b1d eaaa8b5aa464158735be51174f1f0a02d6b07e1fa50b46e7c58e23812c107
6a0
CLIENT_HANDSHAKE_TRAFFIC_SECRET ce880f05144e0e3afac2b130124585842e547b81caa6a45ef46fc50d22abf7f2 f07aa48bb341276af12a92294f41a0bfd82810161572c
f77650ba061b6fa8586
SERVER_HANDSHAKE_TRAFFIC_SECRET ce880f05144e0e3afac2b130124585842e547b81caa6a45ef46fc50d22abf7f2 5ffda0ed8fc0067ab0607be10585812f9a41be178648a
8ead3258bb61cd5b6ae
CLIENT_TRAFFIC_SECRET_0 ce880f05144e0e3afac2b130124585842e547b81caa6a45ef46fc50d22abf7f2 e2e6256a5bdbc208d10a3c857a3727c7fb6850c98acee872a25e3
98129d674e5
SERVER_TRAFFIC_SECRET_0 ce880f05144e0e3afac2b130124585842e547b81caa6a45ef46fc50d22abf7f2 daf4ea47dbf8cd4f276a16af7e7f26db4ea2c6d37763e174823bb
83d4f9416ea
EXPORTER_SECRET ce880f05144e0e3afac2b130124585842e547b81caa6a45ef46fc50d22abf7f2 9bc68f4f66fd264940ec22a5838a22853f3a4f7db87f28a2853d867385732
da3
CLIENT_HANDSHAKE_TRAFFIC_SECRET 25d3e2d7b335fef55ffc506de180e61fa4299e9f5dbd835baec375907e47aecd 22abd1f69b9fbc1580f033ffa1988b8525979d224e0d4
82af993c1e346c70732
SERVER_HANDSHAKE_TRAFFIC_SECRET 25d3e2d7b335fef55ffc506de180e61fa4299e9f5dbd835baec375907e47aecd 1d835c54039a7a336a77873392b080868f22b29fd6408
```

session key & key negotiation data written in the sslkeylog.txt file

This file can also be fed to Wireshark to inspect the captured encrypted communication.

For better understanding of the content you can refer to Mozilla's documentation NSS Key Log Format.

AV Variant:
Eric observed that the Chrome browser installed on his machine gave him a warning. This warning informs a user that the system configuration has instructed it to leak the secret keys.



On further investigation Eric found that Avast was using a browser extension to set SSLKEYLOGFILE variable in Chrome's process memory which was pointing to a named pipe.

Pointing SSLKEYLOGFILE to a named pipe rather than a local file indicates that all the session keys were sent to another process, in this case the Avast AV engine.

Impact

Having access to the encryption keys of TLS sessions gives one an ability to decrypt the encrypted traffic. Many Antivirus engines use this technique to inspect the encrypted traffic for possible threat or exfiltration.

The same can be misused by third parties to spying, surveillance, snooping posing serious threat to once privacy and personal security.

Mitigation

Users need to depend on browsers to inform them of any attempt of accessing session keys by extensions. Till the time browsers decide to make this feature mainstream, users need to be vigilant of what they are installing on their system.

References

- https://twitter.com/ericlaw
- https://textslashplain.com/2019/08/11/spying-on-https/
- https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format

2.6.7 Breaking PDF Encryptions

The Portable Document Format (PDF) is a file format that can have the following as its contents: text, fonts, vector graphics, raster images. The PDF specification also provides for encryption and digital signatures, file attachments, and metadata to enable workflows requiring these features. Businesses find many uses for encrypted PDFs for example Banks use them for confidentiality when exchanging documents with customers, Medical diagnostic devices use secure PDFs to send test results to patients or medics. Government agencies accept incoming documents as encrypted PDFs.

Many companies use encrypted PDFs to transfer data through an unsecured or untrusted channel — for example, to upload a file to cloud storage that many people have access to.

Introduction

PDF encryption is a concept where a PDF file is encrypted and only someone who has the password can see what's in the file. According to the file format's specifications, PDF supports encryption, using the AES algorithm with Cipher Block Chaining encryption mode.

The two attack scenarios that let third parties to gain access to the encrypted content. Direct exfiltration does not require any special cryptography skills, only an understanding of the PDF format specifications.

The second attack, called a Malleability attack, is more complicated and requires an understanding of Cipher Block Chaining mode.

Description

Direct exfiltration: The idea of this attack is to abuse the partial encryption feature by modifying an encrypted PDF file. As soon as the file is opened and decrypted by the victim sensitive content is sent to the attacker. Encrypted PDF files do not have integrity protection. Thus, an attacker can modify the structure of encrypted PDF documents, add unencrypted objects, or wrap encrypted parts into a context controlled by the attacker.

The most relevant object for the attack is the definition of an *Action*, which can submit a form, invoke a URL, or execute JavaScript. The execution of the *Action* can be

triggered automatically once the PDF file is opened (after the decryption) or via user interaction, for example, by clicking within the document.

This attack has three requirements to be successful. While all requirements are PDF standard compliant, they have not necessarily been implemented by every PDF application:

- Partial encryption: Partially encrypted documents based on Crypt Filters like the *Identity* filter or based on other less supported methods like the *None* encryption algorithm.
- Cross-object references: It must be possible to reference and access encrypted string or stream objects from unencrypted attacker-controlled parts of the PDF document.
- Exfiltration channel: One of the interactive features allowing the PDF reader to communicate via the Internet must exist, with or without user interaction. Such Features are *PDF Forms*, *Hyperlinks*, or *JavaScript*.

Malleability attack: This attack on PDF encryption employs a known drawback of Cipher Block Chaining (CBC) mode, which lacks integrity control. The essence of this well-known attack is that an attacker who knows part of the plain-text information that was encrypted can change the contents of a block.

However, according to the PDF format specifications, each time content in a PDF file is encrypted, it also encrypts different permissions. This was done to prevent attackers from tampering with permissions, which are encrypted with the same AES key as the rest of the document.

The permissions are stored in the file in an unencrypted form by default attackers know what 12 bytes of the file are, and as a result, they can tamper with Cipher Block Chaining to target and manipulate encrypted data, for example, adding the data exfiltration mechanism to the encrypted file to send the contents of the file to a third-party site.

This attack has two necessary preconditions:

- ***Known plaintext*:** To manipulate an encrypted object using CBC gadgets, a known-plaintext segment is necessary. For AESV3 – the most recent encryption algorithm – this plain- text is always given by the Perms entry. For older versions, known plaintext from the object to be exfiltrated is necessary.
- ***Exfiltration channel*:** One of the interactive features: *PDF Forms* or *Hyperlinks*.

Exploitation

Attack 1.1: Exfiltration via PDF Forms
The PDF standard allows a document's encrypted streams or strings to be defined as values of a PDF form to be submitted to an external server. This can be done by referencing their object numbers as the values of the form fields within the *Catalog* object. To make the form auto-submit itself once the document is opened and decrypted, an *OpenAction* can be applied. Note that the object which contains the URL *(http://p.df)* for form submission is not encrypted and completely controlled by the attacker. As a result, as soon as the victim opens the PDF file and decrypts it, the *OpenAction* will be executed by sending the decrypted content to *(http://p.df)*.

Attack 1.2: Exfiltration via Hyperlinks:
The PDF standard allows setting a "base" URI in the *Catalog* object used to resolve all relative URIs in the document. This enables an attacker to define the encrypted part as a relative URI to be leaked to the attacker's web server. Therefore the base URI will be prepended to each URI called within the PDF file. In the given example, we set the base URI to *(http://p.df)*.

The plaintext can be leaked by clicking on a visible element such as a link, or without user interaction by defining a URI *Action* to be automatically performed once the document is opened.

Attack 1.3: Exfiltration via JavaScript
The PDF JavaScript reference allows JavaScript code within a PDF document to directly access arbitrary string/stream objects within the document and leak them with functions such as *getDataObjectContents* or *getAnnots*. With a JavaScript action that is automatically triggered once the document is opened. The attack has some advantages compared to Exfiltration via PDF Forms and Exfiltration via Hyperlinks, such as the flexibility of an actual programming language.

Attack 2.1: Exfiltration via PDF Forms
PDF allows the submission of string and stream objects to a web server. This can be used in conjunction with CBC gadgets to leak the plaintext to an attacker-controlled server, even if partial encryption is not allowed.

A CBC gadget constructed from the known plaintext can be used as the submission URL. The construction of this particular URL gadget is challenging. As PDF encryption uses PKCS#5 padding, constructing the URL using a single gadget from the known *Perms* plaintext is difficult, as the last 4 bytes that would need to contain the padding are unknown.

Two techniques to solve this. we can take the last block of an unknown ciphertext and append it to our constructed URL, essentially reusing the correct PKCS#5 padding of the unknown plaintext. Unfortunately, this would introduce 20 bytes of random data from the gadgeting process and up to 15 bytes of the unknown plaintext to the end of our URL.

The other technique uses PDF standard that allows the execution of multiple OpenActions in a document, allowing us to essentially guess the last padding byte of the Perms value. This is possible by iterating over all 256 possible values of the last plaintext byte to get 0x01, resulting in a URL with as little random as possible (3 bytes). As a limitation, if one of the 3 random bytes contains special characters, the form submission URL might break.

Attack 2.2: Exfiltration via Hyperlinks
An attacker can construct URLs in the encrypted PDF document that contains the plaintext to exfiltrate. However, it does not require the setting of a “base” URI in plaintext to achieve exfiltration. The constructed URL contains random bytes from the gadgeting process, which may prevent the exfiltration in some cases.

Attack 2.3: Exfiltration via Half-Open Object Streams
An attacker can use *ObjectStreams* which allow the storage of arbitrary objects inside a stream. The attacker uses an object stream to define new objects using CBC gadgets. An object stream always starts with a header of space-separated integers that define the object number and the byte offset of the object inside the stream. The dictionary of an object stream contains the key First which defines the byte offset of the first object inside the stream. An attacker can use this value to create a comment of arbitrary size by setting it to the first byte after their comment.

Impact

Breaking the encryption of PDF leads to Sensitive information disclosure. Most of the PDF readers were vulnerable to this attack.

- PDFKit is vulnerable - CVE-2019-8772
- PDF-XChange Editor/Viewer is vulnerable
- Master PDF Editor
- Adobe Acrobat and Reader versions 2019.012.20035 and earlier, 2017.011.30142 and earlier, 2017.011.30143 and earlier, 2015.006.30497 and earlier, and 2015.006.30498.

Mitigation

- PDF format must encrypt the entire content, This encryption makes it difficult for an attacker to include their content into the file.
- More secure encryption algorithms must be used.
- Must not have backward compatibility of the PDF version.

References

- https://www.pdf-insecurity.org/encryption/encryption.html
- https://www.kaspersky.co.in/blog/36c3-pdf-encryption/19448/
- https://i.blackhat.com/eu-19/Thursday/eu-19-Muller-How-To-Break-PDF-Encryption-2.pd

3. Appendix

Below is the list of few more vulnerabilities that we have captured but haven't covered in the white papers.

Sr.No	Vulnerability	Description
1	A research paper demonstrates the use of thermal laser stimulation for a fault-based attack on the memory of field-programmable gate arrays (FPGAs)	A vulnerability describes key extraction using Thermal Laser simulation technique. The full description of the vulnerability is given in the below link. https://eprint.iacr.org/2018/717
2	Problem with timestamps in eIDAS digital certificates, which goes against Estonian law.	The vulnerability describes the validity of vast majority of esonian digital signature scheme in question. The full description of the vulnerability is given in the below link. https://cybersec.ee/timesign/
3	56K broken RSA keys in IoT devices	The vulnerability describes risk of RSA keys when improper random number generation is used in creation of public and private keys of RSA keys and as these certificates are used in IoT devices containing sensitive data, this has serious impacts. The full description of the vulnerability is given in the below link. https://info.keyfactor.com/factoring-rsa-keys-in-the-iot-era#methodology

Sr.No	Vulnerability	Description
4	Compromised key for a certificate from Sectigo & other insecure practices	Private key used in a TLS certificate was compromised and was made available online. Sectigo failed to revoke the certificate immediately showing the lack of concern for basic CA best practices and its commitment towards extending ethical and considerate service expected for a public Certifying Authority https://twitter.com/tomwas54/status/1162114413148725248 Sectigo issued certificates with concerningly mismatched subject information https://groups.google.com/g/mozilla.dev.security.policy/c/Cd8PtS3vtvU?pli=1