X.509DoS Attack

X.509DoS: Exploiting and Detecting Denial-of-Service Vulnerabilities in Cryptographic Libraries using Crafted X.509 Certificates

Bing Shi[1], Wenchao Li[1], Yuchen Wang[1], Xiaolong Bai[1], Luyi Xing[2]

[1]Alibaba Group, [2]Indiana University Bloomington

34th USENIX Security Symposium

Motivation & Introduction

- Existing research on cryptographic vulnerabilities mainly focuses on confidentiality or integrity, with limited attention paid to **availability**.

Motivation & Introduction

- Existing research on cryptographic vulnerabilities mainly focuses on confidentiality or integrity, with limited attention paid to **availability**.
- To fill this gap, we conducted a comprehensive study targeting implementations vulnerable to DoS (Denial-of-Service) attacks within cryptographic libraries.

Motivation & Introduction

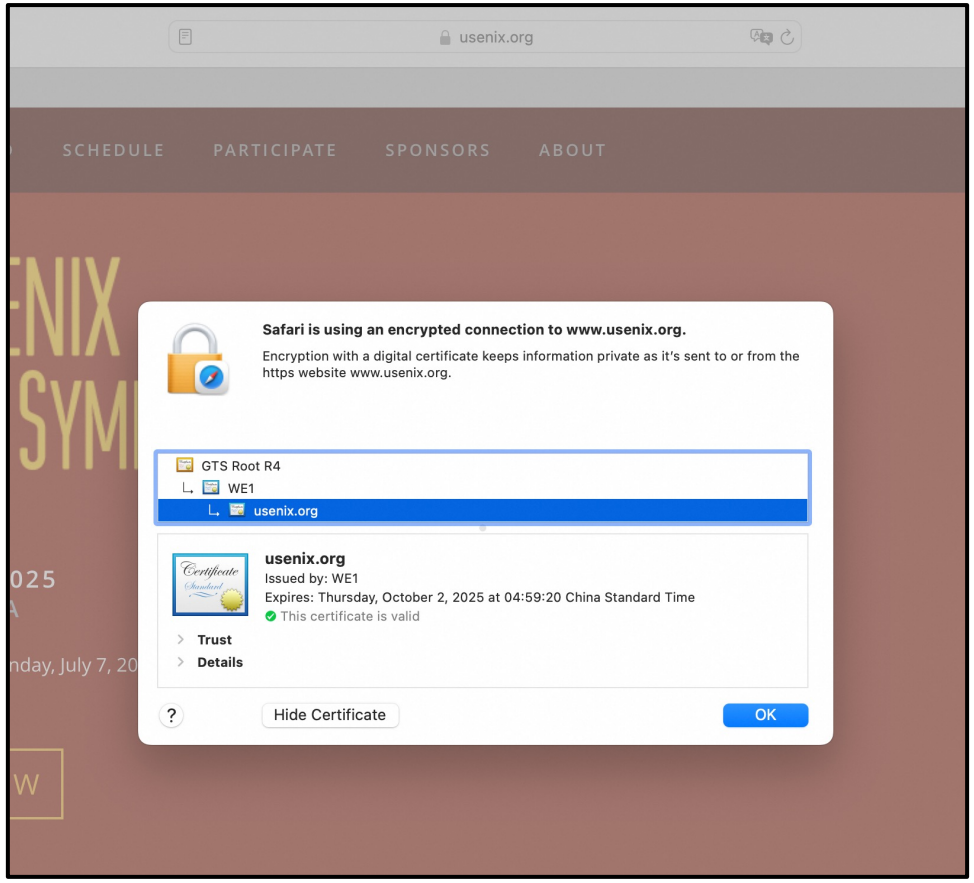
- Existing research on cryptographic vulnerabilities mainly focuses on confidentiality or integrity, with limited attention paid to **availability**.
- To fill this gap, we conducted a comprehensive study targeting implementations vulnerable to DoS (Denial-of-Service) attacks within cryptographic libraries.
- Notably, we observed that these vulnerable implementations were frequently associated, directly or indirectly, with X.509 certificates.

Motivation & Introduction

- Existing research on cryptographic vulnerabilities mainly focuses on confidentiality or integrity, with limited attention paid to **availability**.
- To fill this gap, we conducted a comprehensive study targeting implementations vulnerable to DoS (Denial-of-Service) attacks within cryptographic libraries.
- Notably, we observed that these vulnerable implementations were frequently associated, directly or indirectly, with X.509 certificates.
- Consequently, we facilitated the launch of DoS attacks by using crafted X.509 certificates as attack vectors, which we termed **X.509DoS** in this work.

Background

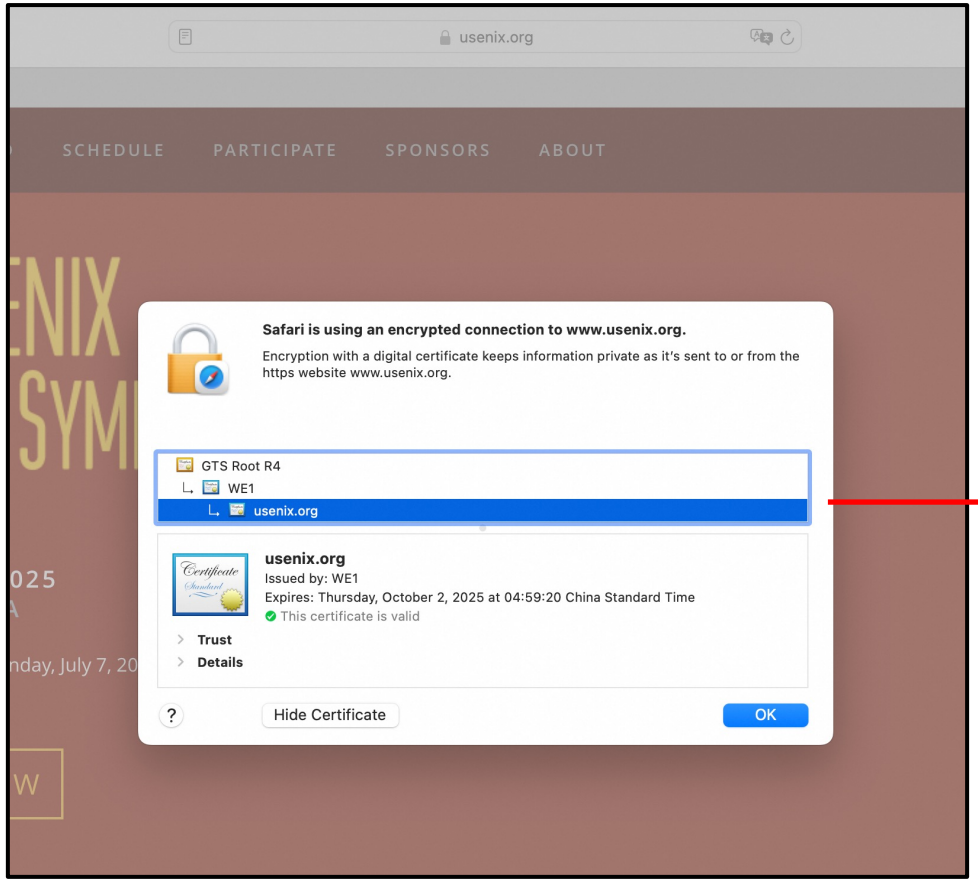
- X.509 is a widely adopted standard for PKI, defining the format of digital certificates used across various scenarios.



The official website of USENIX Security

Background

- X.509 is a widely adopted standard for PKI, defining the format of digital certificates used across various scenarios.



The official website of USENIX Security

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            1f:d2:38:da:4e:a2:60:25:0e:c9:ec:4d:18:bc:a9:8a
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: C=US, O=Google Trust Services, CN=WE1
        Validity
            Not Before: Jul  3 20:00:53 2025 GMT
            Not After : Oct  1 20:59:20 2025 GMT
        Subject: CN=usenix.org
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:e6:3e:d3:54:19:5d:fa:6d:ae:c4:3a:6e:5c:0e:
                    b1:32:6a:3b:3c:37:39:33:5f:cb:8b:ad:f8:23:c8:
                    ...
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
                ...
    Signature Algorithm: ecdsa-with-SHA256
    Signature Value:
        30:46:02:21:00:8b:40:60:85:01:f4:56:fe:aa:54:df:3c:b1:
        2d:fd:98:53:4b:da:9f:58:9a:c8:71:9f:97:de:63:86:9b:a5:
        ...
```

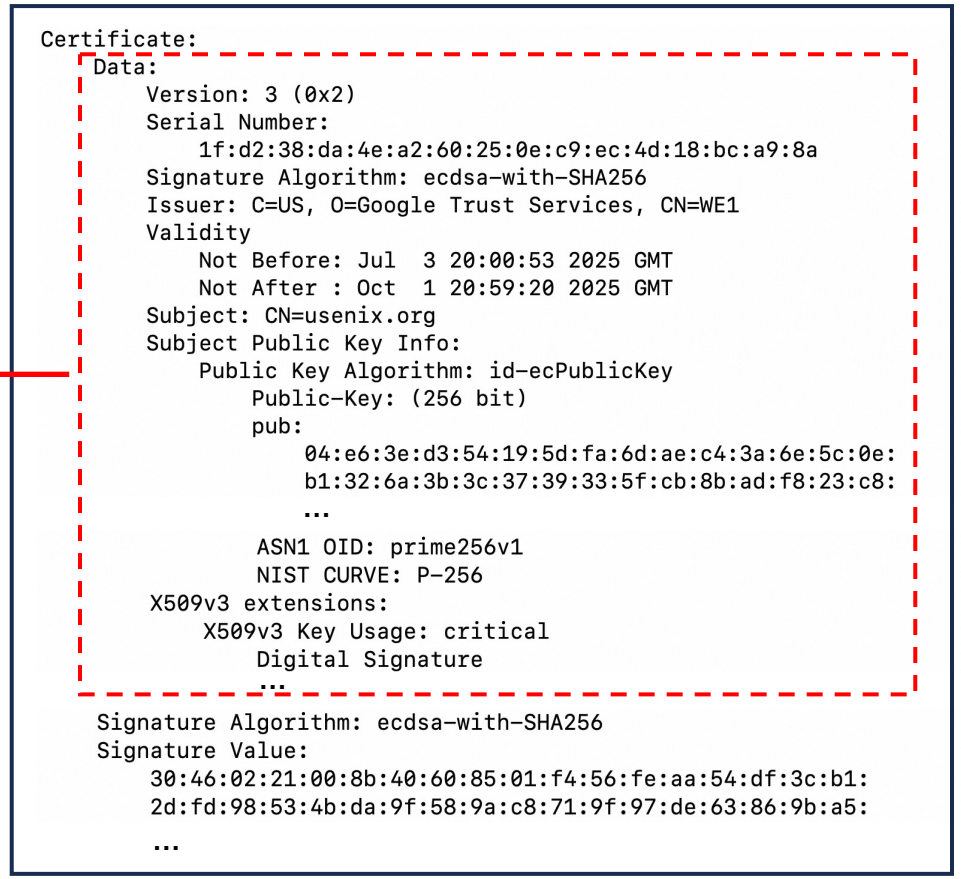
The certificate of usenix.org

Background

- X.509 is a widely adopted standard for PKI, defining the format of digital certificates used across various scenarios.

tbsCertificate

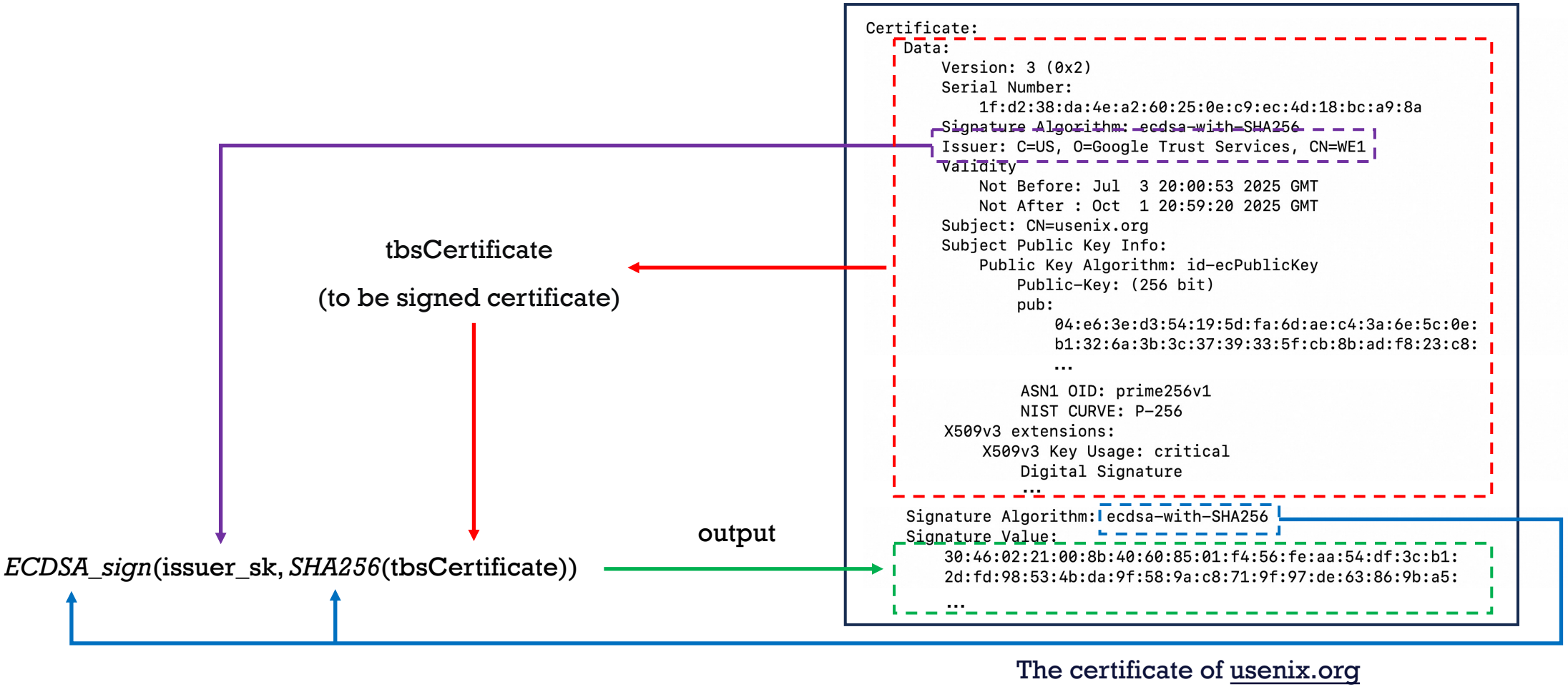
(to be signed certificate)



The certificate of usenix.org

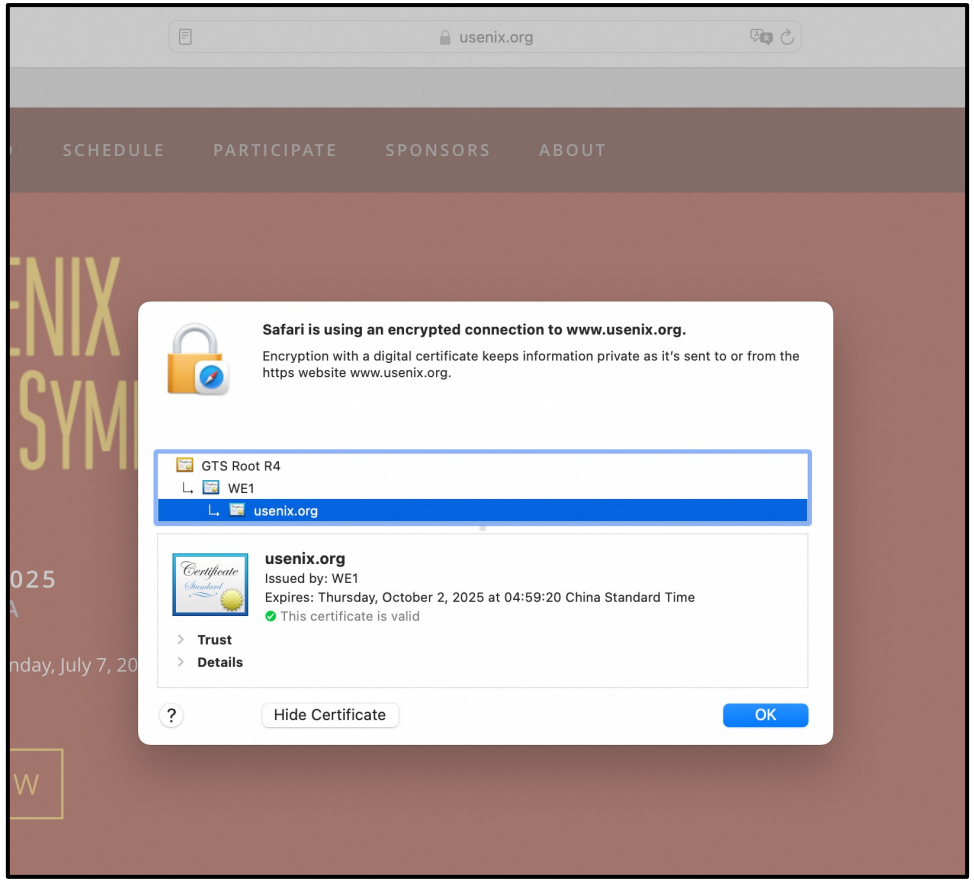
Background

- X.509 is a widely adopted standard for PKI, defining the format of digital certificates used across various scenarios.

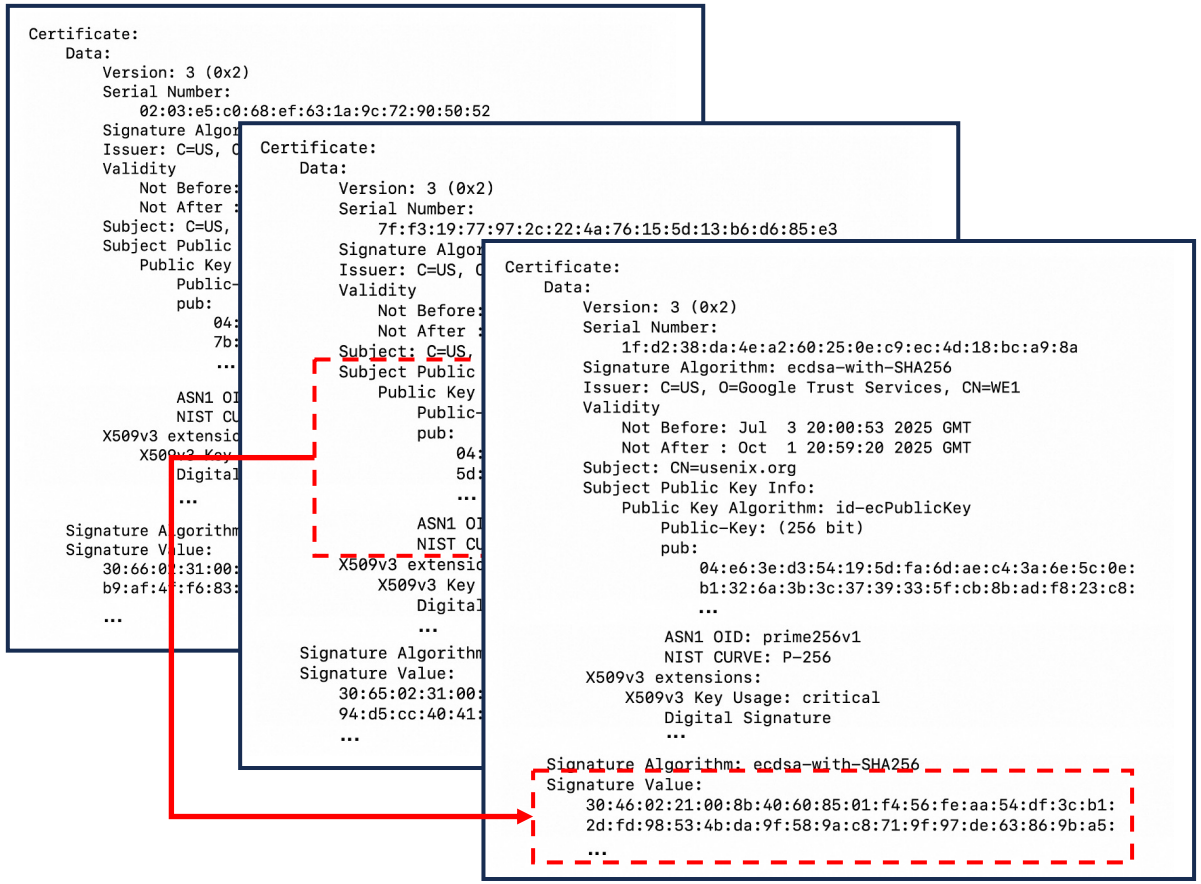


Background

- X.509 is a widely adopted standard for PKI, defining the format of digital certificates used across various scenarios.



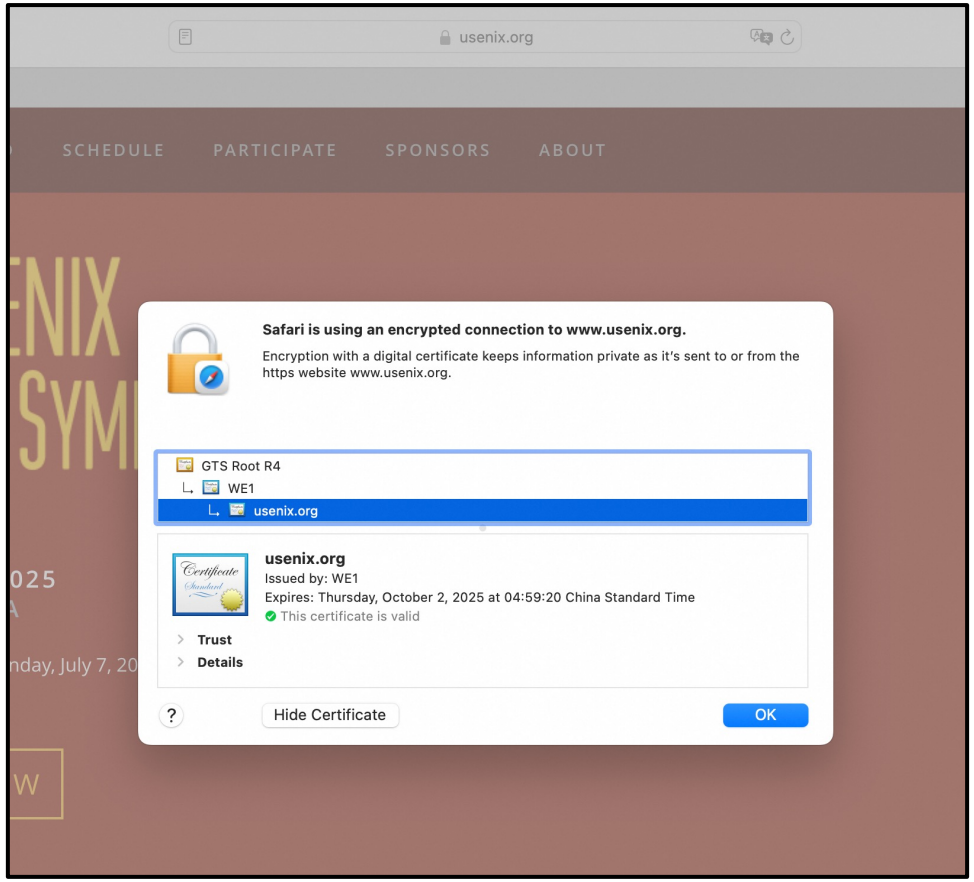
The official website of USENIX Security



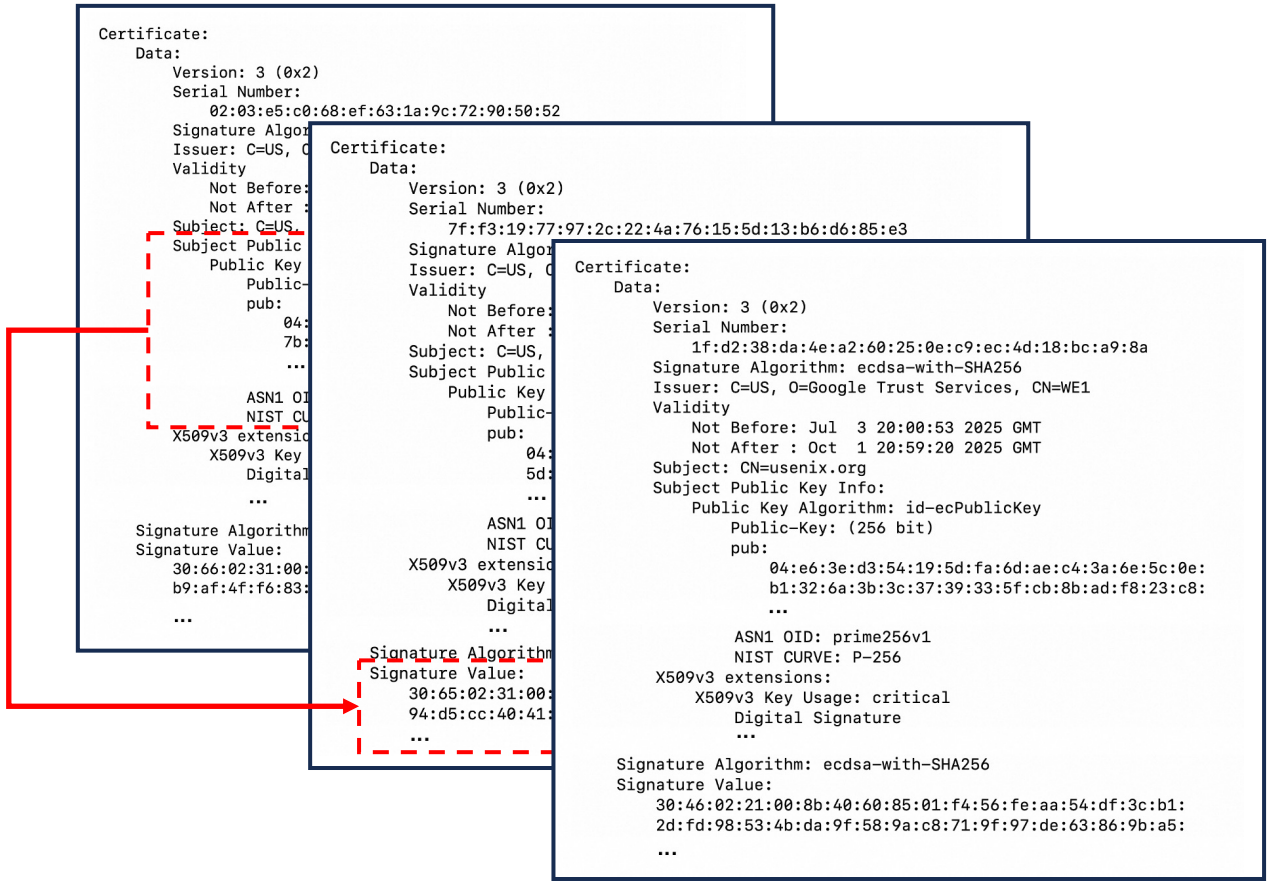
Verify using the issuer's public key

Background

- X.509 is a widely adopted standard for PKI, defining the format of digital certificates used across various scenarios.



The official website of USENIX Security



Verify using the issuer's public key

Security Analysis & Exploitation

- We conducted a comprehensive study targeting implementations vulnerable to DoS attacks within cryptographic libraries, with a detailed analysis of 10 different types of DoS risks that arose in the **Math**, **ASN.1**, and **X.509** modules.

Table 1: An overview of the 10 DoS risks analyzed in this work

Risk-ID	Module	Impact	Behavior	Flaw
1	Math	I/II	Performing arithmetic operations in $\mathbb{F}_{2^m}$	No limit on the size of m
2	Math	III	Performing reduction in $\mathbb{F}_{2^m}$	No check on the order of the indices
3	Math	I	Computing square roots in $\mathbb{F}_p$	No check on the primality of p
4	Math	I	Checking the primality of p	No limit on the size of p
5	ASN.1	I	Decoding ASN.1 objects in structured types	No limit on the number of elements
6	ASN.1	II	Decoding ASN.1 objects	No limit on the size of the *Length* field
7	ASN.1	I/II	Decoding OBJECT IDENTIFIERs	No limit on the size of the sub-identifiers
8	X.509	I	Validating names during path validation	No limit on the number of names / constraints
9	X.509	I/II	Building the policy tree during path validation	No limit on the number of nodes
10	X.509	I/II	Building the path during path validation	No detection of the cycle in a certificate chain

I: CPU Exhaustion II: Memory Exhaustion III: Crash

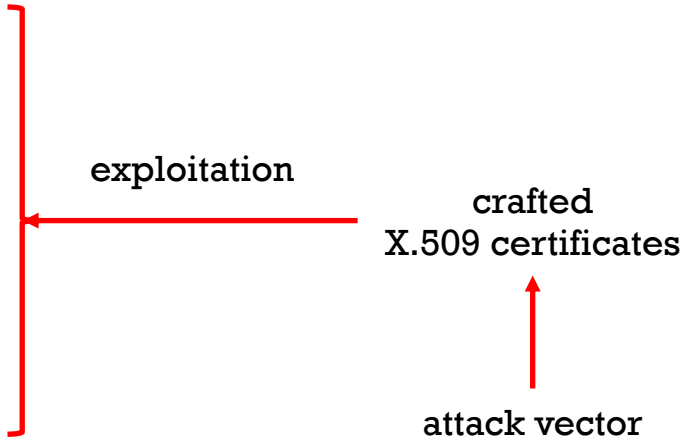
Security Analysis & Exploitation

- We conducted a comprehensive study targeting implementations vulnerable to DoS attacks within cryptographic libraries, with a detailed analysis of 10 different types of DoS risks that arose in the **Math**, **ASN.1**, and **X.509** modules.
- We show how a unified attack vector — **the crafted X.509 certificate** — can be leveraged to exploit each of these risks.

Table 1: An overview of the 10 DoS risks analyzed in this work

Risk-ID	Module	Impact	Behavior	Flaw
1	Math	I/II	Performing arithmetic operations in $\mathbb{F}_{2^m}$	No limit on the size of m
2	Math	III	Performing reduction in $\mathbb{F}_{2^m}$	No check on the order of the indices
3	Math	I	Computing square roots in $\mathbb{F}_p$	No check on the primality of p
4	Math	I	Checking the primality of p	No limit on the size of p
5	ASN.1	I	Decoding ASN.1 objects in structured types	No limit on the number of elements
6	ASN.1	II	Decoding ASN.1 objects	No limit on the size of the *Length* field
7	ASN.1	I/II	Decoding OBJECT IDENTIFIERs	No limit on the size of the sub-identifiers
8	X.509	I	Validating names during path validation	No limit on the number of names / constraints
9	X.509	I/II	Building the policy tree during path validation	No limit on the number of nodes
10	X.509	I/II	Building the path during path validation	No detection of the cycle in a certificate chain

I: CPU Exhaustion II: Memory Exhaustion III: Crash



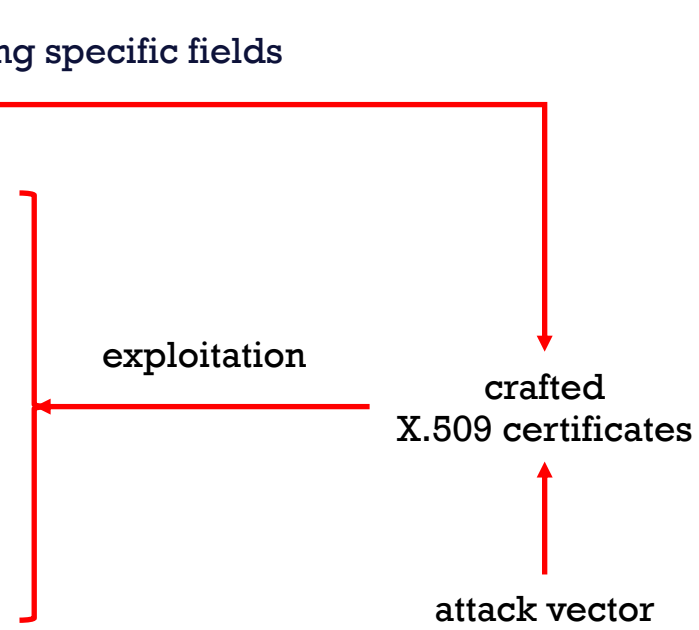
Security Analysis & Exploitation

- We conducted a comprehensive study targeting implementations vulnerable to DoS attacks within cryptographic libraries, with a detailed analysis of 10 different types of DoS risks that arose in the **Math**, **ASN.1**, and **X.509** modules.
- We show how a unified attack vector — **the crafted X.509 certificate** — can be leveraged to exploit each of these risks.
- Exploitation of these risks occurs during the certificate parsing or path validation, which typically precedes signature verification. This allows an attacker to modify the certificate without requiring the CA's private key for re-signing. 😈

Table 1: An overview of the 10 DoS risks analyzed in this work

Risk-ID	Module	Impact	Behavior	Flaw
1	Math	I/II	Performing arithmetic operations in $\mathbb{F}_{2^m}$	No limit on the size of m
2	Math	III	Performing reduction in $\mathbb{F}_{2^m}$	No check on the order of the indices
3	Math	I	Computing square roots in $\mathbb{F}_p$	No check on the primality of p
4	Math	I	Checking the primality of p	No limit on the size of p
5	ASN.1	I	Decoding ASN.1 objects in structured types	No limit on the number of elements
6	ASN.1	II	Decoding ASN.1 objects	No limit on the size of the *Length* field
7	ASN.1	I/II	Decoding OBJECT IDENTIFIERs	No limit on the size of the sub-identifiers
8	X.509	I	Validating names during path validation	No limit on the number of names / constraints
9	X.509	I/II	Building the policy tree during path validation	No limit on the number of nodes
10	X.509	I/II	Building the path during path validation	No detection of the cycle in a certificate chain

I: CPU Exhaustion II: Memory Exhaustion III: Crash



Tool Development & Experiments

- We developed ***X.509DoSTool***, a tool to enable rapid generation of crafted certificates and automatic detection of DoS vulnerabilities in cryptographic libraries.

Artifact Evaluation

Tool Development & Experiments

- We developed ***X.509DoSTool***, a tool to enable rapid generation of crafted certificates and automatic detection of DoS vulnerabilities in cryptographic libraries.
- Leveraging this tool, we successfully discovered 18 new vulnerabilities across six widely used cryptographic libraries.

Table 2: Statistical results of risks across cryptographic libraries by using crafted X.509 certificates

	Risk-1	*Risk-2*	*Risk-3*	*Risk-4*	*Risk-5*	*Risk-6*	*Risk-7*	*Risk-8*	*Risk-9*	*Risk-10*
OpenSSL	✘	✘	✔	✔	✘	✔	✔	✔	✔	✘
Botan	\	\	✔	✔	✔	✘	✘	✔	\	✔
Bouncy Castle	✔	✘	✔	✘	✘	✘	✔	✔	✘	✘
Crypto++	✔	✔	✔	✘	✘	✔	✘	\	\	\
GnuTLS	\	\	\	\	✔	✘	✘	✔	✘	✔
phpseclib	✔	✘	✔	✔	✘	✘	✔	\	\	\
Security (Apple)	\	\	\	\	✔	✘	✔	✔	✔	✘

✔: Detected (Newly Discovered) ✔: Detected (Previously Known) ✘: Mitigated \: Relevant implementations are absent

Responsible Disclosure

CVE-2023-27560
CVE-2023-49316
CVE-2023-50980
CVE-2023-50981
CVE-2024-12133
CVE-2024-12243
CVE-2024-27354
CVE-2024-27355
CVE-2024-29857
CVE-2024-34702
CVE-2024-34703
CVE-2024-54538
...

Tool Development & Experiments

- We developed ***X.509DoSTool***, a tool to enable rapid generation of crafted certificates and automatic detection of DoS vulnerabilities in cryptographic libraries.
- Leveraging this tool, we successfully discovered 18 new vulnerabilities across six widely used cryptographic libraries.
- We selected two vulnerabilities as case studies to further show the threats of the X.509DoS attack in real-world scenarios.

Table 2: Statistical results of risks across cryptographic libraries by using crafted X.509 certificates

	Risk-1	*Risk-2*	*Risk-3*	*Risk-4*	*Risk-5*	*Risk-6*	*Risk-7*	*Risk-8*	*Risk-9*	*Risk-10*
OpenSSL	✘	✘	✔	✔	✘	✔	✔	✔	✔	✘
Botan	\	\	✔	✔	✔	✘	✘	✔	\	✔
Bouncy Castle	✔	✘	✔	✘	✘	✘	✔	✔	✘	✘
Crypto++	✔	✔	✔	✘	✘	✔	✘	\	\	\
GnuTLS	\	\	\	\	✔	✘	✘	✔	✘	✔
phpseclib	✔	✘	✔	✔	✘	✘	✔	\	\	\
Security (Apple)	\	\	\	\	✔	✘	✔	✔	✔	✘

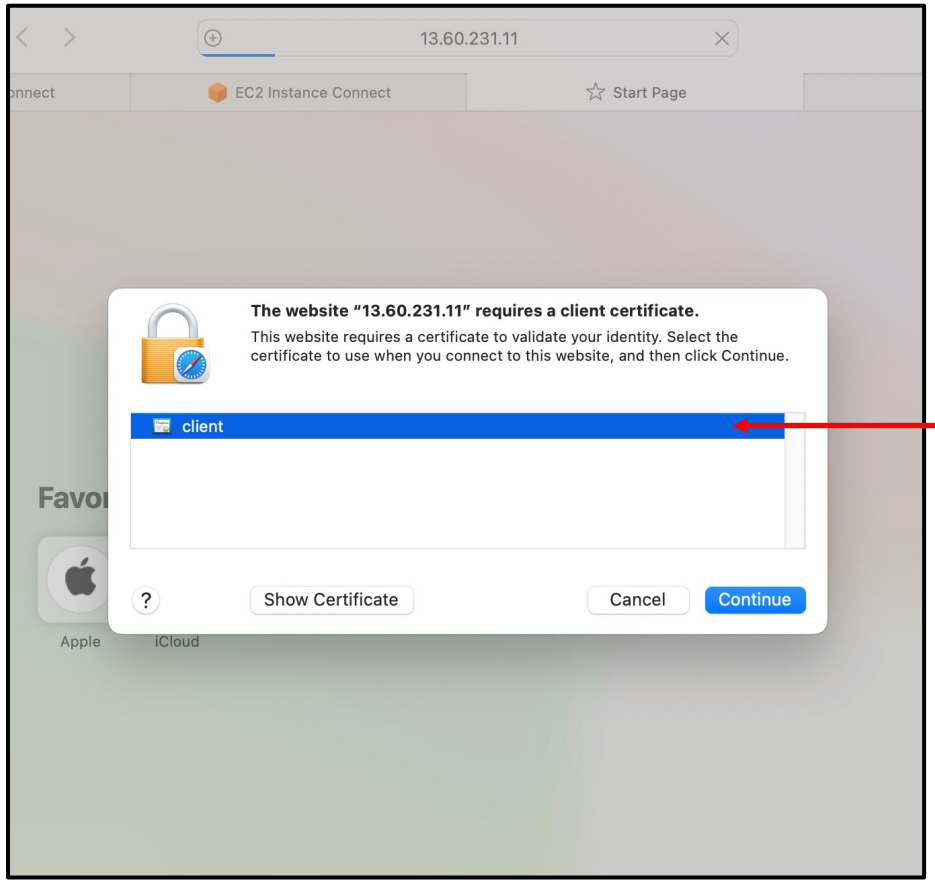
✔: Detected (Newly Discovered) ✔: Detected (Previously Known) ✘: Mitigated \: Relevant implementations are absent

Responsible Disclosure

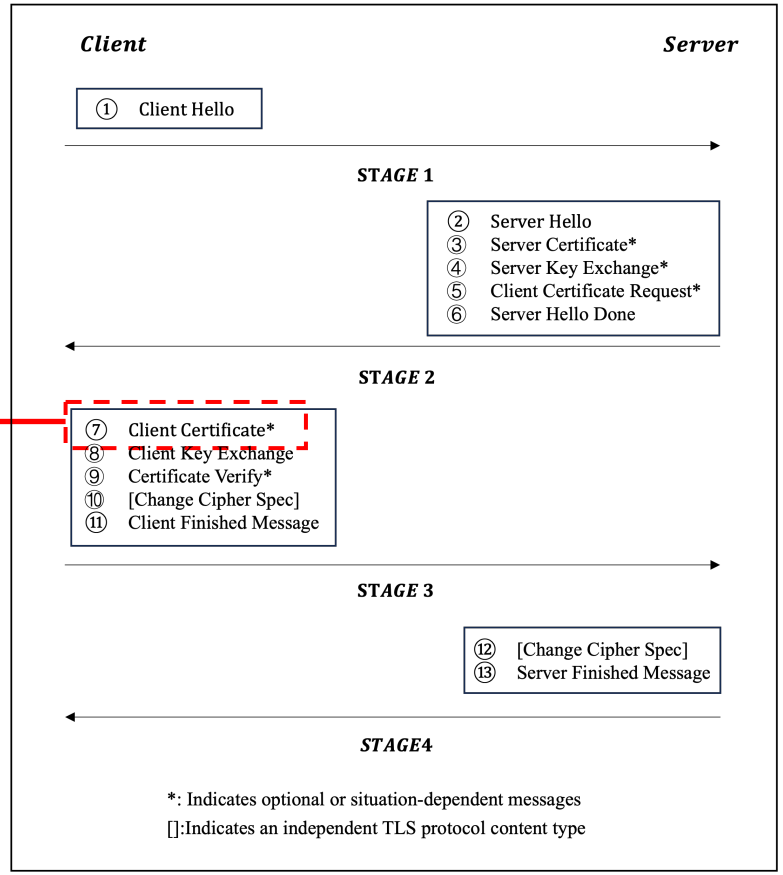
CVE-2023-27560
CVE-2023-49316
CVE-2023-50980
CVE-2023-50981
CVE-2024-12133
CVE-2024-12243
CVE-2024-27354
CVE-2024-27355
CVE-2024-29857
CVE-2024-34702
CVE-2024-34703
CVE-2024-54538

Case Study 1

- Threat Model: (Mutual) TLS Handshake



attack vector

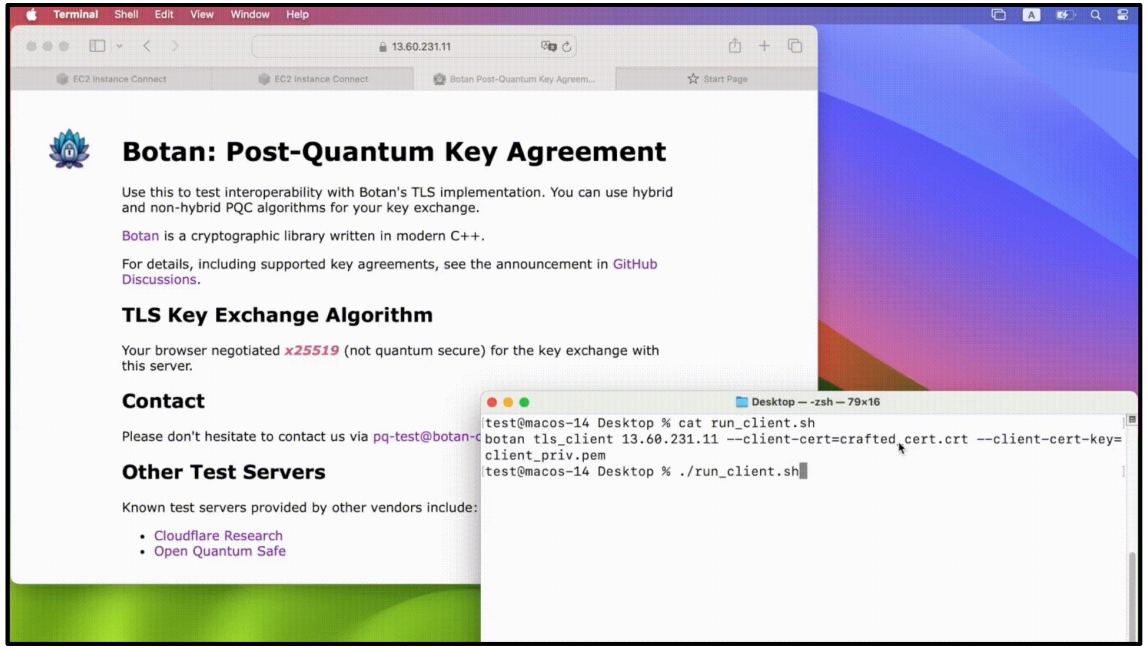


Access a website with mutual TLS enabled

An overview of the TLS v1.2 handshake

Case Study 1

- Experiment: Attacking a website with mutual TLS enabled (using Botan)

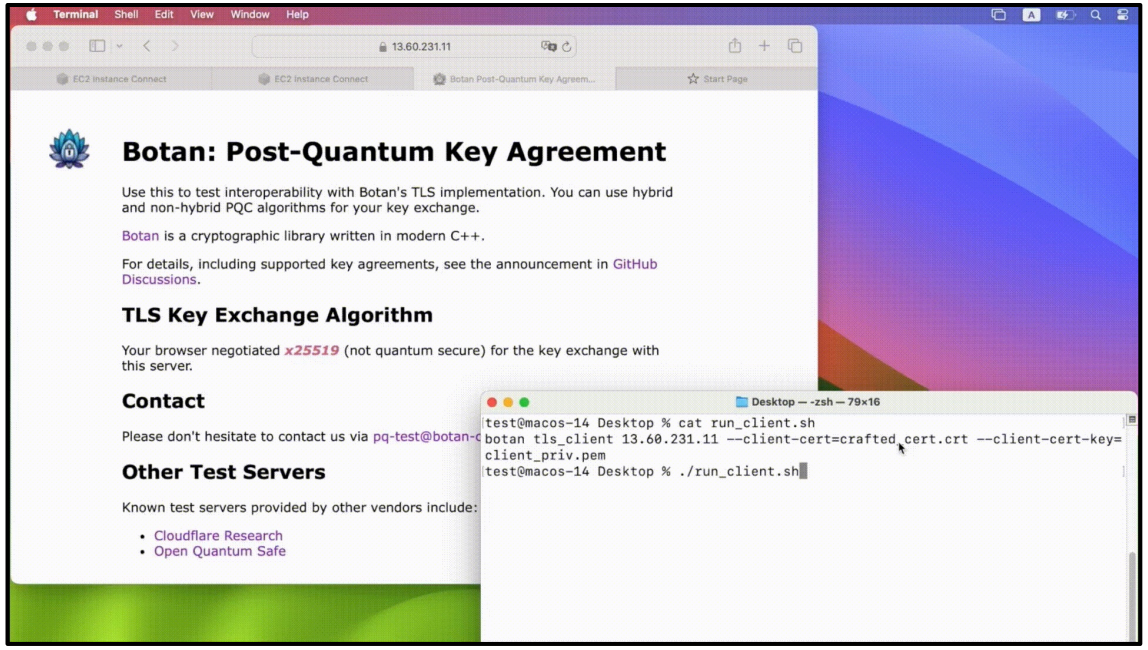


parsing this certificate caused the server's CPU utilization to reach 100%

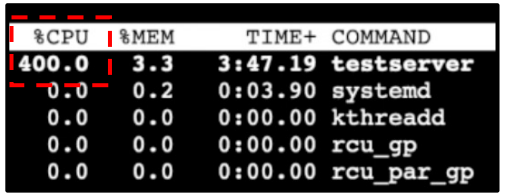
an attacker could impersonate a malicious client and send a crafted certificate to the server

Case Study 1

- Experiment: Attacking a website with mutual TLS enabled (using Botan)



resend



%CPU	%MEM	TIME+	COMMAND
400.0	3.3	3:47.19	testserver
0.0	0.2	0:03.90	systemd
0.0	0.0	0:00.00	kthreadd
0.0	0.0	0:00.00	rcu_gp
0.0	0.0	0:00.00	rcu_par_gp

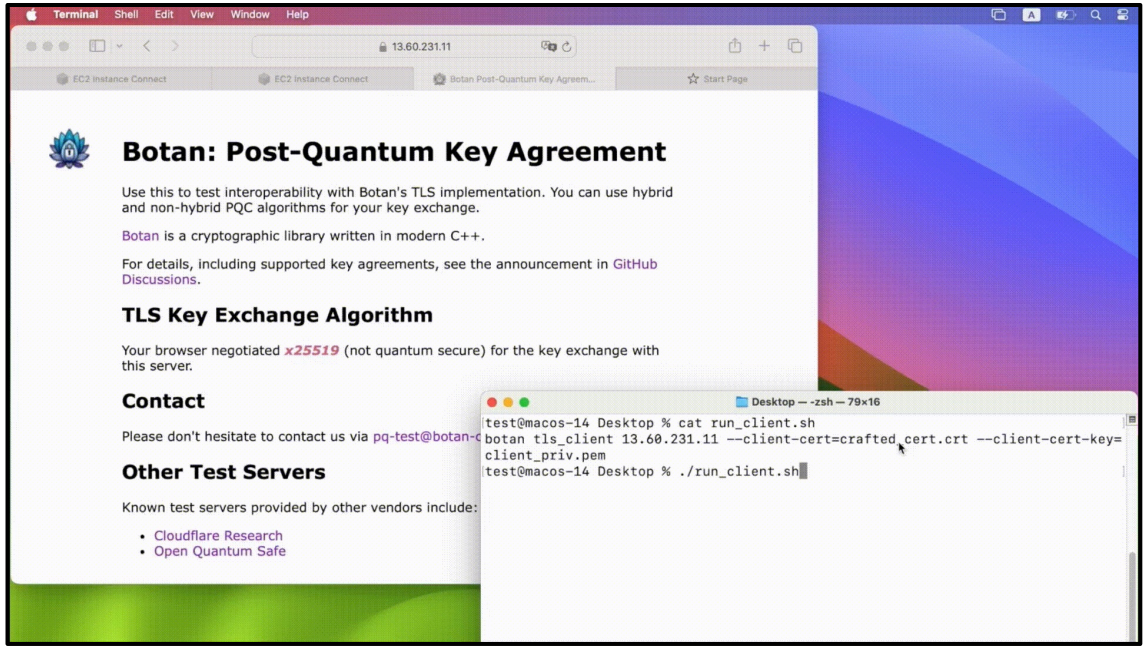
repeating the above process caused each CPU core's utilization to reach 100%

parsing this certificate caused the server's CPU utilization to reach 100%

an attacker could impersonate a malicious client and send a crafted certificate to the server

Case Study 1

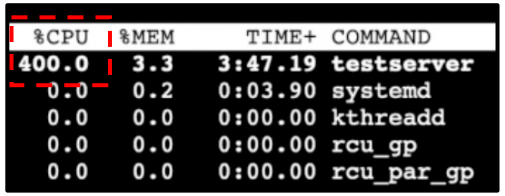
- Experiment: Attacking a website with mutual TLS enabled (using Botan)



parsing this certificate caused the server's CPU utilization to reach 100%

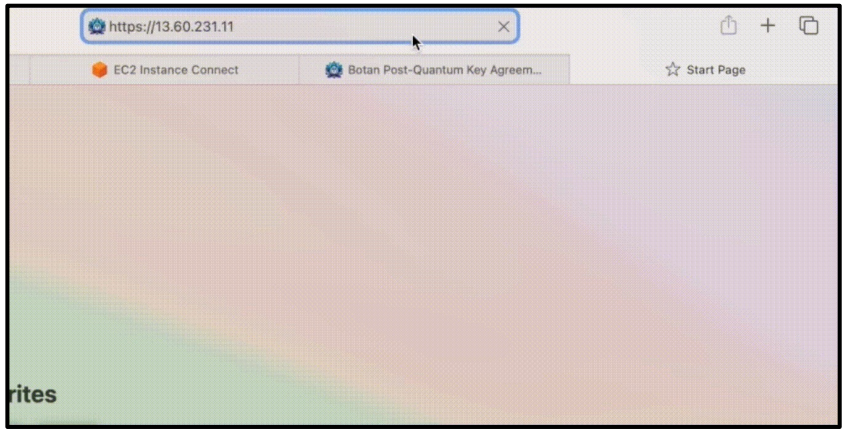
an attacker could impersonate a malicious client and send a crafted certificate to the server

resend



%CPU	%MEM	TIME+	COMMAND
400.0	3.3	3:47.19	testserver
0.0	0.2	0:03.90	systemd
0.0	0.0	0:00.00	kthreadd
0.0	0.0	0:00.00	rcu_gp
0.0	0.0	0:00.00	rcu_par_gp

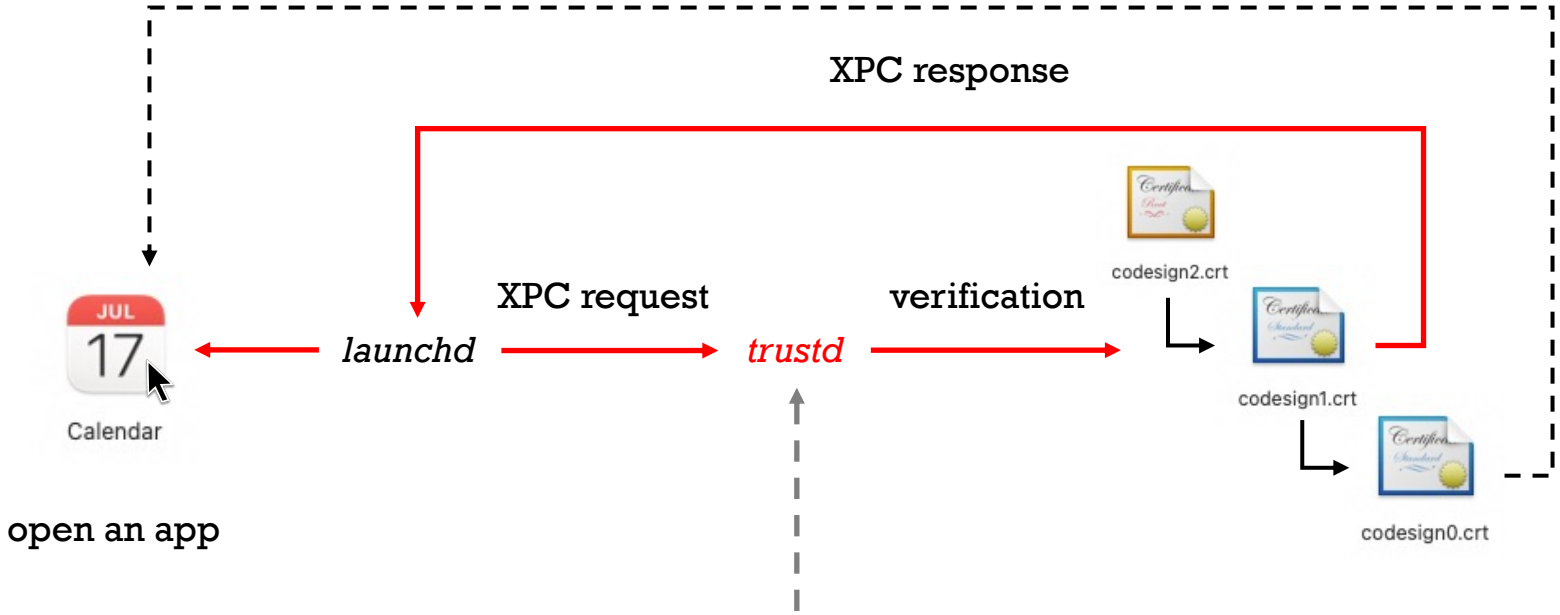
repeating the above process caused each CPU core's utilization to reach 100%



the website is no longer accessible to other users

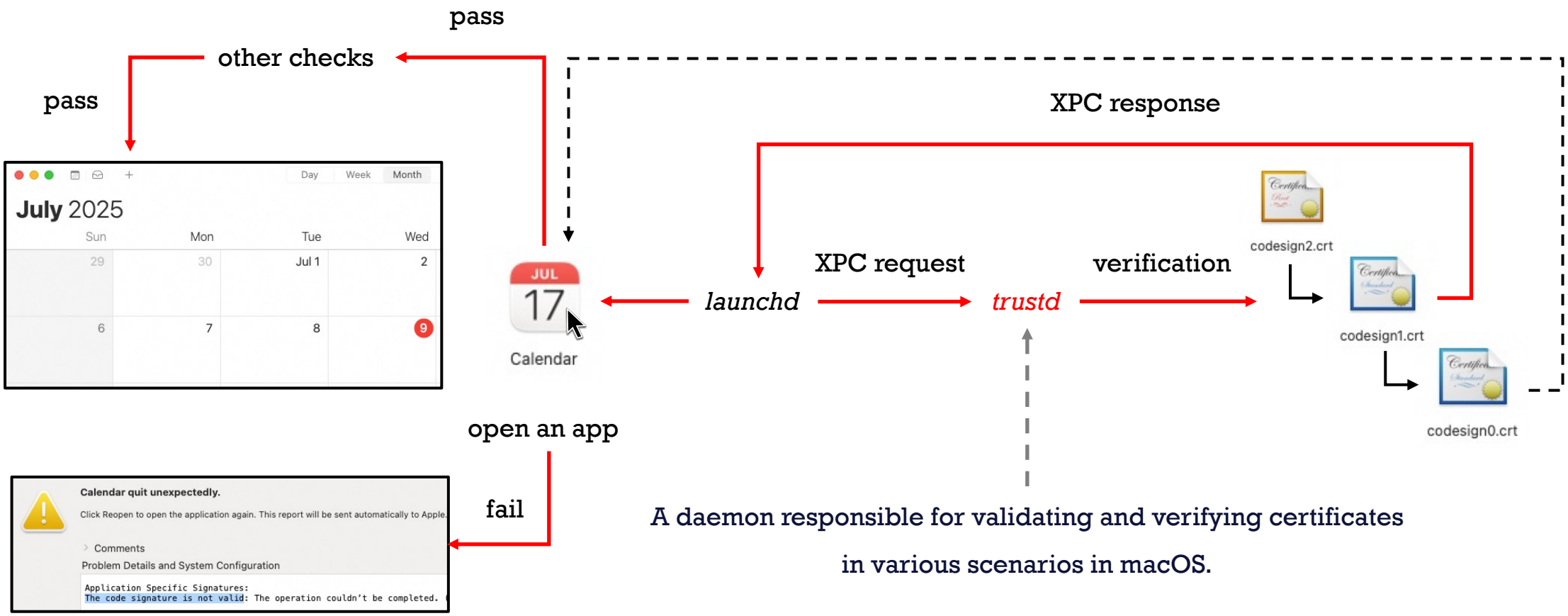
Case Study 2

- Threat Model: App Verification in Operating Systems (e.g., Apple macOS)



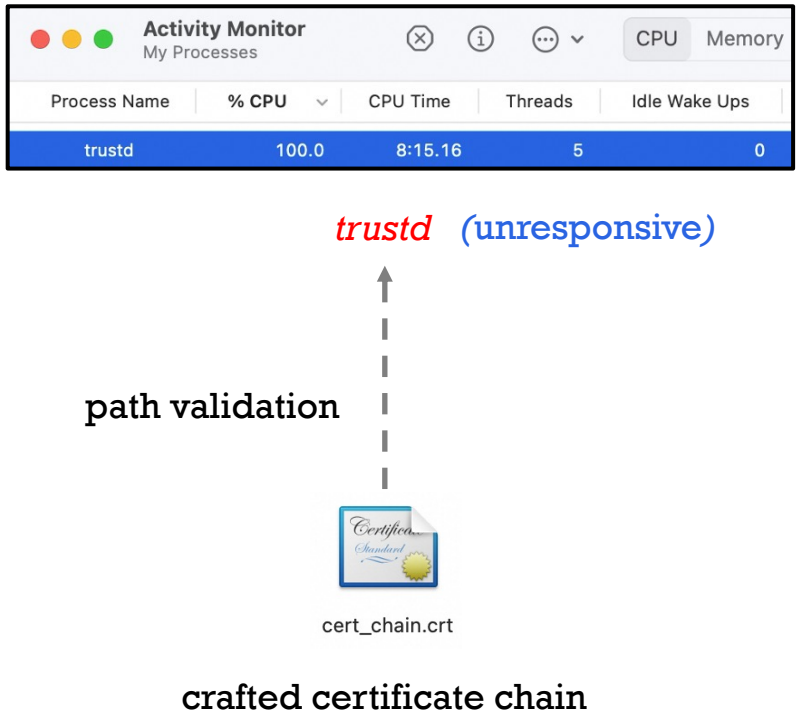
Case Study 2

- Threat Model: App Verification in Operating Systems (e.g., Apple macOS)



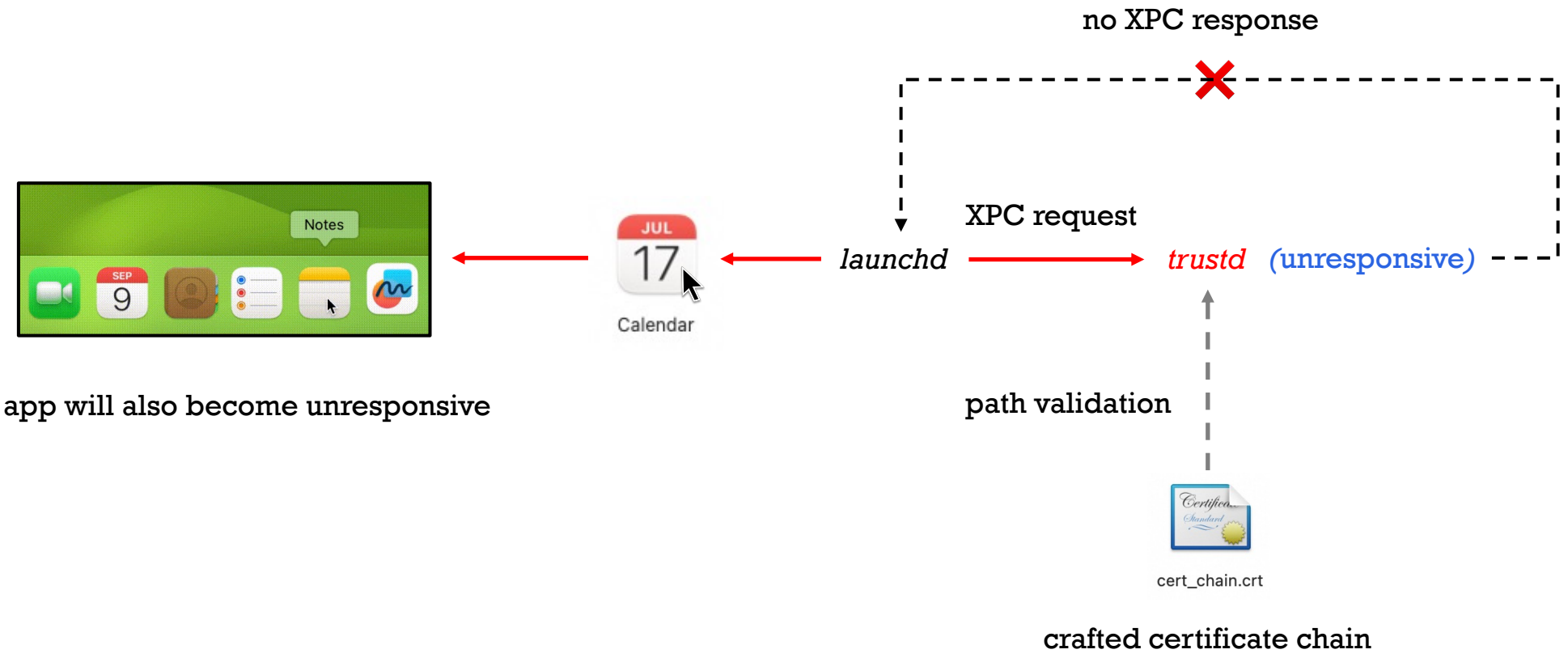
Case Study 2

- Threat Model: App Verification in Operating Systems (e.g., Apple macOS)



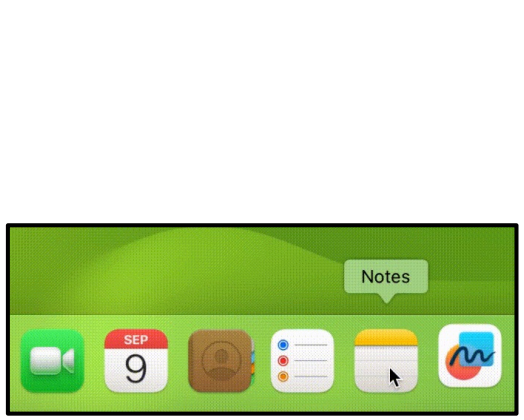
Case Study 2

- Threat Model: App Verification in Operating Systems (e.g., Apple macOS)



Case Study 2

- Threat Model: App Verification in Operating Systems (e.g., Apple macOS)



the app will also become unresponsive

no XPC response

XPC request

launchd → *trustd* (unresponsive)

path validation

how to trigger the validation of such a certificate chain on the target user's device?

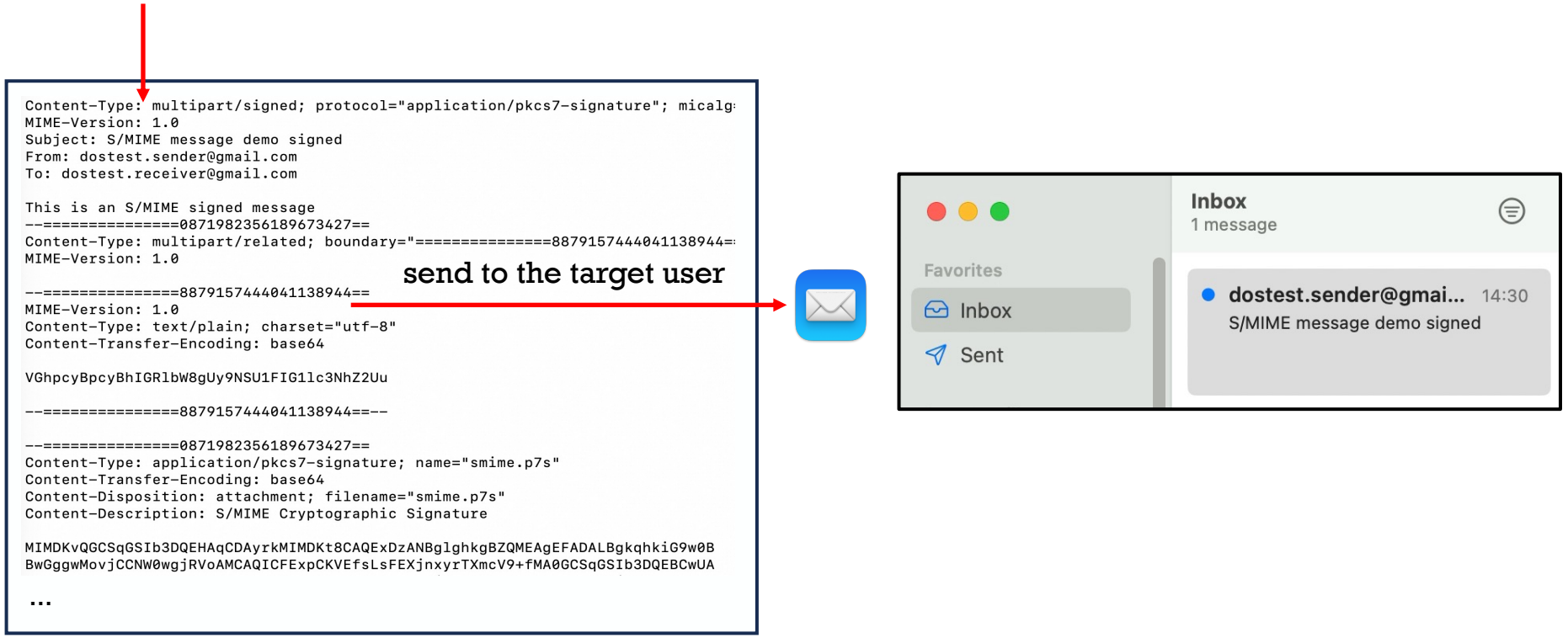
cert_chain.crt

crafted certificate chain

Case Study 2

- Experiment: Attacking a target user's macOS via S/MIME email

sign_message(msg, KEY, CERT, additional_certs = CA_CERTS)



S/MIME mail with crafted certificate chain

Case Study 2

- Experiment: Attacking a target user's macOS via S/MIME email

sign_message(msg, KEY, CERT, additional_certs = CA_CERTS)

```
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg
MIME-Version: 1.0
Subject: S/MIME message demo signed
From: dostest.sender@gmail.com
To: dostest.receiver@gmail.com

This is an S/MIME signed message
----------------0871982356189673427==
Content-Type: multipart/related; boundary="----------------8879157444041138944=
MIME-Version: 1.0

----------------8879157444041138944==
MIME-Version: 1.0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: base64

VGhpcyBpcyBhIGRlbW8gUy9NSU1FIG1lc3NhZ2Uu

----------------8879157444041138944====

----------------0871982356189673427==
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature

MIMDKvQGCSqGSIb3DQEHAqCDAyrkMIMDKt8CAQExDzANBglghkgBZQMEAgEFADALBgkqhkiG9w0B
BwGggwMovjCCNW0wgjRVoAMCAQICFExpCKVEfsLsFEXjnxyrTXmcV9+fMA0GCSqGSIb3DQEBCwUA
...
```

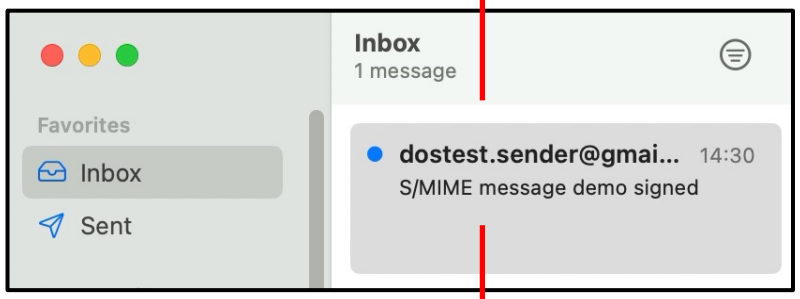
send to the target user

even no need for the user to read this email

S/MIME mail with crafted certificate chain

the *Mail* app trigger the validation and cause *trustd* 100% CPU usage

0-click



0-click

the *Mail* app adds the sender's crafted certificate chain to the recipient's *Keychain*

(~/Library/Keychains/login.keychain-db)

Case Study 2

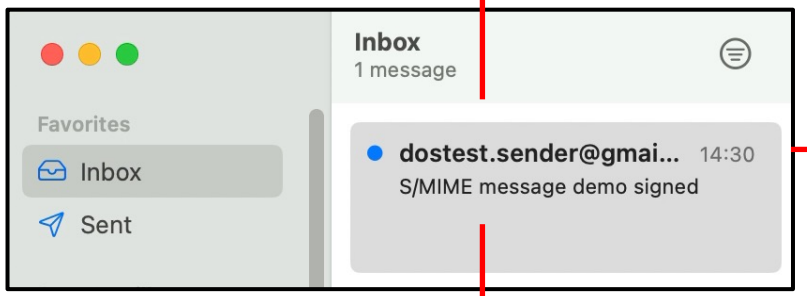
- Experiment: Attacking a target user's macOS via S/MIME email

sign_message(msg, KEY, CERT, additional_certs = CA_CERTS)

```
Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg
MIME-Version: 1.0
Subject: S/MIME message demo signed
From: dostest.sender@gmail.com
To: dostest.receiver@gmail.com

This is an S/MIME signed message
----------------=8719821356189673427==
Content-Type: multipart/related; boundary="----------------8879157444041138944=
MIME-Version: 1.0

----------------=8879157444041138944==
MIME-Version: 1.0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: base64

VGhpcyBpcyBhIGRlbW8gUy9NSU1FIG1lc3NhZ2Uu

----------------=8879157444041138944==--

----------------=8719821356189673427==
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature

MIMDKvQGCSqGSIb3DQEHAqCDAyrkMIMDKt8CAQExDzANBglghkgBZQMEAgEFADALBgkqhkiG9w0B
BwGggwMovjCCNW0wgjRVoAMCAQICFExpCKVEfsLsFEXjnxyrTXmcV9+fMA0GCSqGSIb3DQEBCwUA
...
```

send to the target user

even no need for the user to read this email



0-click

the mail is stored in the *Mail's* inbox

the *Mail* app trigger the validation and cause *trustd* 100% CPU usage

the attack can survive reboots

0-click

the certificates are stored in the *Keychain*

the *Mail* app adds the sender's crafted certificate chain to the recipient's *Keychain*

(~/Library/Keychains/login.keychain-db)

S/MIME mail with crafted certificate chain

Thank you!

paper

website

Contact: Bing Shi

Mail: roadicing@gmail.com