**PDFSage Inc. | Superior Court of Massachusetts, Middlesex County**

**COMMONWEALTH OF MASSACHUSETTS**

**MIDDLESEX, SS.**

**SUPERIOR COURT**

――――――――――――――――――――――――――――――――――

**BO SHANG,**

Plaintiff,

v.

**MIDDLESEX COUNTY DISTRICT**

**ATTORNEY’S OFFICE,**

Defendant.

――――――――――――――――――――――――――――――――――

**COMPLAINT AND JURY DEMAND (AS ENHANCED)**

Plaintiff, Bo Shang (“Plaintiff”), brings this Complaint against Defendant Middlesex

County District Attorney’s Office (“Defendant”) and alleges as follows, incorporating

additional factual and legal authorities:

**1 Plaintiff is an individual residing in 10 McCafferty Way, Burlington MA 01803.**

**2 Defendant is a public office located in Middlesex County, Massachusetts.**

**JURISDICTION AND VENUE**

**3 This Court has subject matter jurisdiction pursuant to G.L. c. 212, § 4, and under**

concurrent jurisdiction principles for claims brought under 42 U.S.C. § 1983. See

Haywood v. Drown, 556 U.S. 729 (2009). This Court also has jurisdiction over claims

arising under the Massachusetts Constitution, the Massachusetts Civil Rights Act

(MCRA), G.L. c. 12, §§ 11H & 11I, and other Massachusetts common law claims.

**4 Venue is proper in this Court pursuant to G.L. c. 223, § 1, because the events or**

omissions giving rise to this action occurred in Middlesex County and because

Defendant is located in Middlesex County.

**FACTUAL BACKGROUND**

**5 On or about January 8, 2025, Plaintiff alleges that the Middlesex District Attorney’s**

Office of MA, described by Plaintiff as “corrupt and despicably morally principled,”

filed a data request to Apple, supposedly under Massachusetts Rules of Civil

Procedure 45.

**6 This occurred on the same day Plaintiff filed a motion to dismiss Twitch and an AirTag +**

commerce tax (App Store) lawsuit against Apple (which hosts the Twitch app).

**7 Plaintiff asserts that this data request was not legitimately obtained under Mass. R. Civ.**

P. 45 but was instead an illegal measure taken against Plaintiff as an “enemy

combatant,” contrary to both domestic and international law, including Geneva

Conventions III & IV, and the International Covenant on Civil and Political Rights

**(ICCPR).**

7A. The United States is a party to the four Geneva Conventions of 1949, which set forth

standards for treatment of persons in armed conflicts, including alleged “enemy

combatants.” Plaintiff maintains that labeling Plaintiff as an “enemy combatant”

without due process violates customary international humanitarian law and Supreme

Court precedent concerning the rights of such individuals. See, e.g., Hamdi v.

Rumsfeld, 542 U.S. 507 (2004); Rasul v. Bush, 542 U.S. 466 (2004); Boumediene v.

Bush, 553 U.S. 723 (2008).

7B. The United States is also a State Party to the ICCPR, which, under Article 9, protects

against arbitrary arrest or detention and, under Article 14, protects due process rights.

Plaintiff alleges that classifying Plaintiff as an “enemy combatant” in a civilian context,

and thereby circumventing ordinary legal process, violates the ICCPR’s guarantees of

fundamental procedural protections.

7C. The Supreme Court has further clarified the rights of individuals designated as “enemy

combatants” in Padilla v. Rumsfeld, 542 U.S. 426 (2004), emphasizing the need for

proper legal process. Plaintiff alleges these precedents reinforce the argument that

civilian processes cannot be bypassed via “enemy combatant” designations.

7D. In Ex parte Milligan, 71 U.S. (4 Wall.) 2 (1866), the Supreme Court held that applying

military or martial process to civilians, when civil courts are open, is unconstitutional.

Plaintiff contends this principle applies here, making any civilian “enemy combatant”

label unlawful.

7E. The Supreme Court in Hamdan v. Rumsfeld, 548 U.S. 557 (2006), further confirmed

that efforts to circumvent civilian courts through alternative proceedings for alleged

combatants violate U.S. constitutional principles. Plaintiff alleges that all such

precedents collectively prohibit unilateral “enemy combatant” branding in non-war

contexts.

**8 On January 30, 2025, Plaintiff received an email from Apple regarding this request,**

which stated in part:

“Apple

**NOTE: THIS NOTICE IS BEING SENT FROM A NO-REPLY EMAIL ACCOUNT—ANY RESPONSE**

**TO THIS EMAIL WILL NOT RECEIVE A RESPONSE**

Dear Account Holder/Customer:

On 2025-01-08, Apple Inc. (“Apple”) received a legal request from Middlesex District

Attorney's Office requesting information regarding your Apple account.

The contact information in relation to the request:

Requesting Agency: Middlesex District Attorney's Office

Requesting Agency Location: Woburn, MA - Massachusetts

Requesting Agency Case Number: 2024-398

Legal Request Type: Subpoena / Summons

Pursuant to the applicable Terms of Service and Apple’s Privacy Policy,

http://www.apple.com/legal/privacy/en-ww/, and as required by U.S. law, Apple

will be producing the requested data in a timely manner as required by the legal

process. If you have questions about the legal request or the information requested,

please contact the requesting agency.

Sincerely,

Apple Privacy & Law Enforcement Compliance

Apple Inc.”

**9 Plaintiff maintains that Defendant violated Plaintiff’s rights under federal and state law**

by improperly obtaining and misusing personal data. Plaintiff asserts a violation of

privacy rights under G.L. c. 214, § 1B (right against unreasonable, substantial or

serious interference with privacy), Article 14 of the Massachusetts Declaration of

Rights (protection against unreasonable searches and seizures), the Fourth Amendment

to the U.S. Constitution, and international human rights norms including Article 17 of

the ICCPR and Article 12 of the Universal Declaration of Human Rights (UDHR).

9A. The UDHR, though not a binding treaty, informs customary international law and reflects

global human rights standards. Article 12 states that “[n]o one shall be subjected to

arbitrary interference with his privacy,” a principle Plaintiff contends was violated.

9B. The United States is also a State Party to the Convention Against Torture (CAT),

highlighting due process norms. Plaintiff claims that Defendant’s labeling and treatment

of Plaintiff as an “enemy combatant” violate the spirit of these international

commitments.

9C. In United States v. Warshak, 631 F.3d 266 (6th Cir. 2010), the court recognized a

reasonable expectation of privacy in certain electronic communications, requiring

proper legal process for data access. Plaintiff alleges Defendant’s conduct flouts

Warshak’s privacy rationale.

9D. In Kyllo v. United States, 533 U.S. 27 (2001), the Supreme Court held that obtaining

information through technology not otherwise accessible without physical intrusion

implicates the Fourth Amendment. Plaintiff characterizes Defendant’s subpoena or

data request as an analogous overreach.

9E. Under Massachusetts jurisprudence, the Supreme Judicial Court in Commonwealth v.

Augustine, 467 Mass. 230 (2014), recognized strong privacy protections for personal

digital records, requiring heightened procedures for obtaining certain data. Plaintiff

alleges that Defendant’s conduct runs afoul of Augustine’s reasoning.

**10 Plaintiff alleges that, in response to Defendant’s perceived threat, Plaintiff invoked the**

Second Amendment to the U.S. Constitution, as recognized in District of Columbia

v. Heller, 554 U.S. 570 (2008), McDonald v. City of Chicago, 561 U.S. 742 (2010), and

Caetano v. Massachusetts, 577 U.S. 411 (2016). Plaintiff also invokes Article 17 of

the Massachusetts Declaration of Rights, contending these decisions protect an

individual right to bear “arms,” which Plaintiff interprets to include “cyber arms.”

**11 Plaintiff claims to have developed or acquired “cyber arms” by creating advanced**

persistent threats (“APTs”) and by allying with other APTs, including “Salt Typhoon.”

Plaintiff asserts that these “cyber arms” are protected under the Second Amendment

and Article 17 as a form of self-defense.

**12 Plaintiff alleges that Defendant’s conduct in issuing or causing the issuance of a data**

request without valid legal basis constituted an unlawful intrusion upon Plaintiff’s data

privacy, in violation of the Fourth Amendment (as incorporated by Mapp v. Ohio, 367

U.S. 643 (1961), and recognized in Katz v. United States, 389 U.S. 347 (1967), Terry v.

Ohio, 392 U.S. 1 (1968), Carpenter v. United States, 138 S. Ct. 2206 (2018), Riley v.

California, 573 U.S. 373 (2014)), Article 14 of the Massachusetts Declaration of Rights,

the Stored Communications Act (18 U.S.C. §§ 2701–2712), Article 17 of the ICCPR,

and Article 12 of the UDHR.

12A. Plaintiff notes that third-party data requests implicate the “third-party doctrine,” as set

forth in Smith v. Maryland, 442 U.S. 735 (1979). However, Carpenter recognized

limitations when sensitive digital data is at issue. Plaintiff alleges that Defendant’s

conduct violates Carpenter’s narrowing of the third-party doctrine.

12B. Plaintiff further cites Commonwealth v. Gouse, 461 Mass. 787 (2012), for the

proposition that Massachusetts courts often apply heightened scrutiny to searches

involving personal or digital privacy, reinforcing Plaintiff’s claim that Defendant’s

subpoena was invalid or overreaching.

**13 Plaintiff contends that Defendant’s conduct effectively labeled Plaintiff an “enemy**

combatant,” heightening constitutional concerns, implicating Article 5 of the UDHR, and

prompting Plaintiff’s reliance on the Second Amendment and Article 17 to protect

“cyber arms” from confiscation, regulation, or direct infringement.

13A. Plaintiff invokes Hamdan v. Rumsfeld, 548 U.S. 557 (2006), to underscore the illegality

of any extrajudicial designation of “enemy combatant” status. Plaintiff argues that

under both domestic and international law, such designations cannot bypass civilian

jurisdiction in ordinary contexts.

**14 Plaintiff asserts that Defendant’s actions violate customary international law norms**

related to privacy, as recognized by multiple treaties and conventions to which the

United States is a party or signatory, including the ICCPR, and contravene prohibitions

on arbitrary interference under global human rights standards.

14A. The United States is a signatory to the Budapest Convention on Cybercrime, addressing

lawful cooperation in criminal cyber matters. Plaintiff contends that Defendant’s

allegedly improper “cyber” classification and data request contravene the spirit of

privacy protections contemplated by such instruments.

14B. Although the United States has not ratified Additional Protocol I or II to the Geneva

Conventions, Plaintiff argues that certain principles therein reflect customary

international humanitarian law, prohibiting arbitrary or extrajudicial designations

of civilians as combatants.

14C. The United States is also a member of the Organization of American States and is bound

by certain obligations under the American Declaration of the Rights and Duties of Man,

which can inform interpretations of privacy and due process in conjunction with other

international norms.

14D. In addition, N.Y. State Rifle & Pistol Assn. v. Bruen, 597 U.S. \_\_\_ (2022), further

clarified the scope of the Second Amendment right to bear arms. Plaintiff references

Bruen to argue that Defendant’s attempts to limit, seize, or regulate “cyber arms”

are inconsistent with the broad individual right recognized by the Supreme Court.

**CAUSES OF ACTION**

**COUNT I**

(Violation of 42 U.S.C. § 1983)

**15 Plaintiff repeats and re-alleges all preceding paragraphs as though fully set forth herein.**

**16 Defendant, acting under color of state law, allegedly caused the issuance of a subpoena**

or summons without proper legal basis in violation of Plaintiff’s constitutional rights,

including but not limited to the Fourth Amendment right to be free from unreasonable

searches and seizures as recognized in Katz, Terry, Mapp, Carpenter, Riley, and related

precedent.

**17 By issuing or causing this allegedly improper process, Defendant deprived Plaintiff of**

rights secured by the Constitution and laws of the United States, in contravention of

**42 U.S.C. § 1983.**

**COUNT II**

(Violation of Massachusetts Civil Rights Act)

**18 Plaintiff repeats and re-alleges all preceding paragraphs as though fully set forth herein.**

**19 Defendant’s conduct—issuing a data request under color of law without legitimate**

basis—constitutes interference or attempted interference with Plaintiff’s exercise or

enjoyment of rights secured by the Constitutions and laws of the United States and

the Commonwealth, including the right against unreasonable searches (Article 14) and

the right to keep arms (Article 17), by means of threats, intimidation, or coercion, in

violation of G.L. c. 12, §§ 11H & 11I. See Batchelder v. Allied Stores Int’l, Inc.,

388 Mass. 83 (1983); Buster v. George W. Moore, Inc., 438 Mass. 635 (2003);

Commonwealth v. Powell, 459 Mass. 572 (2011).

**20 As a direct and proximate result of Defendant’s actions, Plaintiff has suffered and will**

continue to suffer damages recoverable under the MCRA.

**COUNT III**

(Abuse of Process Under Massachusetts Law)

**21 Plaintiff repeats and re-alleges all preceding paragraphs as though fully set forth herein.**

**22 Under Massachusetts law, an abuse of process claim arises when legal process is used**

for an ulterior or illegitimate purpose. See Cohen v. Hurley, 20 Mass. App. Ct. 439

(1985); Kelley v. Stop & Shop Cos., 26 Mass. App. Ct. 557 (1988); Lorusso v. Bloom,

321 Mass. 9 (1947).

**23 Defendant allegedly misused legal process by pursuing a data request unsupported by**

valid legal grounds and did so for an improper purpose, causing harm to Plaintiff.

**24 As a direct and proximate result of Defendant’s actions, Plaintiff has suffered damages**

recoverable under Massachusetts law.

**COUNT IV**

(Injunctive Relief Under Federal and State Law)

**25 Plaintiff repeats and re-alleges all preceding paragraphs as though fully set forth herein.**

**26 As a result of Defendant’s conduct, Plaintiff seeks injunctive relief prohibiting**

Defendant from further unlawful use of subpoenas, summonses, or other legal process

to access Plaintiff’s personal data without proper justification. Plaintiff seeks to enjoin

any acts by Defendant that violate Plaintiff’s rights under federal and state law,

including the Fourth Amendment, Article 14, G.L. c. 214, § 1B, the MCRA, the Stored

Communications Act, and international human rights treaties such as the ICCPR.

**COUNT V**

(Assertion of the Second Amendment and

Article 17 of the Massachusetts Declaration of Rights)

**27 Plaintiff repeats and re-alleges all preceding paragraphs as though fully set forth herein.**

**28 The Second Amendment states that “the right of the people to keep and bear Arms,**

shall not be infringed.” As held in District of Columbia v. Heller, 554 U.S. 570 (2008),

this right is individual in nature, and in McDonald v. City of Chicago, 561 U.S. 742

(2010), it applies to the states. In Caetano v. Massachusetts, 577 U.S. 411 (2016),

the Supreme Court reiterated its broad scope. Article 17 of the Massachusetts

Declaration of Rights similarly protects the right to keep and bear arms. N.Y. State

Rifle & Pistol Assn. v. Bruen, 597 U.S. \_\_\_ (2022), further refines these constitutional

principles.

**29 Plaintiff asserts that “cyber arms” (i.e., advanced persistent threats, digital tools, or**

alliances with groups such as “Salt Typhoon”) constitute protected “arms” under the

Second Amendment and Article 17. Plaintiff alleges that any attempt by Defendant

to seize, regulate, or otherwise interfere with these “cyber arms” without due process

violates Plaintiff’s federal and state constitutional rights.

**30 Plaintiff further alleges that Defendant’s labeling of Plaintiff as an “enemy combatant”**

or any related act to disarm Plaintiff’s “cyber capacity” contravenes Heller, McDonald,

Caetano, Bruen, and Article 17 of the Massachusetts Declaration of Rights.

**31 Plaintiff therefore seeks declaratory relief that any effort by Defendant to restrict**

Plaintiff’s possession or development of “cyber arms” violates the Second Amendment

and Article 17, and that such restriction contravenes self-defense principles acknowledged

by various human rights instruments, including the UN Charter’s Article 51 (albeit in

state contexts) and related customary international law.

**REQUEST FOR RELIEF**

WHEREFORE, Plaintiff respectfully requests that this Court:

A. Enter judgment in favor of Plaintiff and against Defendant on all causes of action;

B. Award Plaintiff compensatory, consequential, and punitive damages in an amount to be

determined at trial;

**C Grant injunctive relief restraining Defendant from seeking or using Plaintiff’s personal**

data without proper legal justification;

**D Declare that Plaintiff’s “cyber arms” are protected under the Second Amendment and**

Article 17, and that any attempt by Defendant to restrict or confiscate them, if any,

violates federal and state constitutions and relevant international human rights standards;

E. Award Plaintiff’s reasonable attorneys’ fees and costs pursuant to 42 U.S.C. § 1988,

G.L. c. 12, §§ 11H & 11I, or as otherwise provided by law;

F. Grant such other and further relief as the Court deems just and proper.

**DEMAND FOR JURY TRIAL**

Plaintiff demands a trial by jury on all issues so triable.

Dated: 2/27/2025

Respectfully submitted,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Bo Shang

10 McCafferty Way

Burlington MA 01803-3127

202-235-5017 | 781-999-4101

bo@shang.software | enigmatic.typhoon@gmail.com

**EXHIBIT 1: On 1/30/25, the Plaintiff receives an email from Apple detailing the information request made to the Plaintiff’s developer account on 1/8/25, by the Middlsex DA’s Office. This date coincided with the Plaintiff filing 27 and 27-1 in Federal Court Case 3:24-cv-06664-JS, the first time ever anyone has won a Section 230 claim vs Twitch interactive.**

https://www.fakeopenai.co/section230

https://www.fakeopenai.co/lsat

**EXHIBIT 2: The Plaintiff is making great progress, and expects to achieve an “Eternal” family of zero-day capabilities on the SMBv2 protocol, within a day or few days.**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* File: smb2\_pipe\_exec\_client.c ss

\*

\* Demonstrates:

\* 1. Connecting to an SMB2/3 server (TCP 445).

\* 2. Negotiate, Session Setup, Tree Connect to IPC$.

\* 3. Create/open the named pipe "\\PIPE\\svcctl".

\* 4. Partially demonstrate sending a DCERPC bind

\* request to the SVCCTL interface (stub only).

\* 5. Read back any server response.

\* 6. Close the pipe with an SMB2 Close.

\*

\* Security & Production Warnings:

\* - This remains incomplete demonstration code:

\* - No real auth or signing.

\* - No real DCERPC parse/marshalling logic.

\* - Minimal error handling and no encryption.

\* - Use only in a controlled environment with

\* permission!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <stdint.h>

#include <errno.h>

#pragma pack(push, 1)

//--------------------------------------------------

// SMB2 Header

//--------------------------------------------------

typedef struct \_SMB2Header {

unsigned char ProtocolId[4]; // 0xFE 'S' 'M' 'B'

uint16\_t StructureSize; // Always 64 for SMB2

uint16\_t CreditCharge; // Credits requested/charged

uint32\_t Status; // For responses, server sets status

uint16\_t Command; // SMB2 command code

uint16\_t Credits; // Credits granted/requested

uint32\_t Flags; // SMB2 header flags

uint32\_t NextCommand; // Offset to next command in compound

uint64\_t MessageId; // Unique message ID

uint32\_t Reserved; // Usually 0

uint32\_t TreeId; // Tree ID

uint64\_t SessionId; // Session ID

unsigned char Signature[16]; // For signing (unused here)

} SMB2Header;

// SMB2 Commands

#define SMB2\_NEGOTIATE 0x0000

#define SMB2\_SESSION\_SETUP 0x0001

#define SMB2\_TREE\_CONNECT 0x0003

#define SMB2\_CREATE 0x0005

#define SMB2\_CLOSE 0x0006

#define SMB2\_READ 0x0008

#define SMB2\_WRITE 0x0009

// SMB2 Status Codes (common)

#define STATUS\_SUCCESS 0x00000000

#define STATUS\_INVALID\_PARAMETER 0xC000000D

#define STATUS\_ACCESS\_DENIED 0xC0000022

#define STATUS\_NOT\_SUPPORTED 0xC00000BB

// SMB2 Dialects

#define SMB2\_DIALECT\_0202 0x0202

#define SMB2\_DIALECT\_0210 0x0210

#define SMB2\_DIALECT\_0300 0x0300

//--------------------------------------------------

// Minimal Structures for Basic SMB2 Ops

//--------------------------------------------------

**/\* SMB2 NEGOTIATE \*/**

typedef struct \_SMB2NegotiateRequest {

uint16\_t StructureSize; // Must be 36

uint16\_t DialectCount;

uint16\_t SecurityMode;

uint16\_t Reserved;

uint32\_t Capabilities;

uint64\_t ClientGuid; // Simplified to 8 bytes for demonstration

uint32\_t NegotiateContextOffset;

uint16\_t NegotiateContextCount;

uint16\_t Reserved2;

// Then dialect array

} SMB2NegotiateRequest;

typedef struct \_SMB2NegotiateResponse {

uint16\_t StructureSize; // Must be 65 in real SMB2

uint16\_t SecurityMode;

uint16\_t DialectRevision;

uint16\_t NegotiateContextCount;

uint32\_t ServerGuid; // Simplified

uint32\_t Capabilities;

uint32\_t MaxTransSize;

uint32\_t MaxReadSize;

uint32\_t MaxWriteSize;

uint64\_t SystemTime;

uint64\_t ServerStartTime;

// etc...

} SMB2NegotiateResponse;

**/\* SMB2 SESSION\_SETUP \*/**

typedef struct \_SMB2SessionSetupRequest {

uint16\_t StructureSize; // Must be 25

uint8\_t Flags;

uint8\_t SecurityMode;

uint32\_t Capabilities;

uint32\_t Channel;

uint16\_t SecurityBufferOffset;

uint16\_t SecurityBufferLength;

// Security buffer follows...

} SMB2SessionSetupRequest;

typedef struct \_SMB2SessionSetupResponse {

uint16\_t StructureSize; // Must be 9

uint16\_t SessionFlags;

uint16\_t SecurityBufferOffset;

uint16\_t SecurityBufferLength;

// ...

} SMB2SessionSetupResponse;

**/\* SMB2 TREE\_CONNECT \*/**

typedef struct \_SMB2TreeConnectRequest {

uint16\_t StructureSize; // Must be 9

uint16\_t Reserved;

uint32\_t PathOffset;

uint32\_t PathLength;

// Path follows

} SMB2TreeConnectRequest;

typedef struct \_SMB2TreeConnectResponse {

uint16\_t StructureSize; // Must be 16

uint8\_t ShareType;

uint8\_t Reserved;

uint32\_t ShareFlags;

uint32\_t Capabilities;

uint32\_t MaximalAccess;

} SMB2TreeConnectResponse;

**/\* SMB2 CREATE \*/**

typedef struct \_SMB2CreateRequest {

uint16\_t StructureSize; // Must be 57

uint8\_t SecurityFlags;

uint8\_t RequestedOplockLevel;

uint32\_t ImpersonationLevel;

uint64\_t SmbCreateFlags;

uint64\_t Reserved;

uint32\_t DesiredAccess;

uint32\_t FileAttributes;

uint32\_t ShareAccess;

uint32\_t CreateDisposition;

uint32\_t CreateOptions;

uint16\_t NameOffset;

uint16\_t NameLength;

uint32\_t CreateContextsOffset;

uint32\_t CreateContextsLength;

// Filename follows...

} SMB2CreateRequest;

typedef struct \_SMB2CreateResponse {

uint16\_t StructureSize; // Must be 89

uint8\_t OplockLevel;

uint8\_t Flags;

uint32\_t CreateAction;

uint64\_t CreationTime;

uint64\_t LastAccessTime;

uint64\_t LastWriteTime;

uint64\_t ChangeTime;

uint64\_t AllocationSize;

uint64\_t EndofFile;

uint32\_t FileAttributes;

// 16-byte FileId

uint64\_t FileIdPersistent;

uint64\_t FileIdVolatile;

// optional create contexts

} SMB2CreateResponse;

/\* SMB2 WRITE/READ (for the RPC data) \*/

typedef struct \_SMB2WriteRequest {

uint16\_t StructureSize; // Must be 49

uint16\_t DataOffset;

uint32\_t Length;

uint64\_t Offset;

uint64\_t FileIdPersistent;

uint64\_t FileIdVolatile;

uint32\_t Channel;

uint32\_t RemainingBytes;

uint16\_t WriteChannelInfoOffset;

uint16\_t WriteChannelInfoLength;

uint32\_t Flags;

// Then the data

} SMB2WriteRequest;

typedef struct \_SMB2WriteResponse {

uint16\_t StructureSize; // Must be 17

uint16\_t Reserved;

uint32\_t Count;

uint32\_t Remaining;

uint16\_t WriteChannelInfoOffset;

uint16\_t WriteChannelInfoLength;

} SMB2WriteResponse;

typedef struct \_SMB2ReadRequest {

uint16\_t StructureSize; // Must be 49

uint8\_t Padding;

uint8\_t Reserved;

uint32\_t Length;

uint64\_t Offset;

uint64\_t FileIdPersistent;

uint64\_t FileIdVolatile;

uint32\_t MinimumCount;

uint32\_t Channel;

uint32\_t RemainingBytes;

uint16\_t ReadChannelInfoOffset;

uint16\_t ReadChannelInfoLength;

} SMB2ReadRequest;

typedef struct \_SMB2ReadResponse {

uint16\_t StructureSize; // Must be 17

uint8\_t DataOffset;

uint8\_t Reserved;

uint32\_t DataLength;

uint32\_t DataRemaining;

uint32\_t Reserved2;

// data follows

} SMB2ReadResponse;

**/\* SMB2 CLOSE \*/**

typedef struct \_SMB2CloseRequest {

uint16\_t StructureSize; // Must be 24

uint16\_t Flags;

uint32\_t Reserved;

uint64\_t FileIdPersistent;

uint64\_t FileIdVolatile;

} SMB2CloseRequest;

typedef struct \_SMB2CloseResponse {

uint16\_t StructureSize; // Must be 60

uint16\_t Flags;

uint32\_t Reserved;

uint64\_t CreationTime;

uint64\_t LastAccessTime;

uint64\_t LastWriteTime;

uint64\_t ChangeTime;

uint64\_t AllocationSize;

uint64\_t EndOfFile;

uint32\_t FileAttributes;

} SMB2CloseResponse;

#pragma pack(pop)

//--------------------------------------------------

// Global State & Helper Functions

//--------------------------------------------------

static uint64\_t gMessageId = 1;

static uint64\_t gSessionId = 0;

static uint32\_t gTreeId = 0;

static int gSock = -1;

static uint64\_t gPipeFidPersistent = 0;

static uint64\_t gPipeFidVolatile = 0;

/\*

\* sendSMB2Request: send an SMB2 header + payload

\*/

int sendSMB2Request(SMB2Header \*hdr, const void \*payload, size\_t payloadLen) {

ssize\_t sent = send(gSock, hdr, sizeof(SMB2Header), 0);

if (sent < 0) {

perror("send header");

return -1;

}

if (payload && payloadLen > 0) {

sent = send(gSock, payload, payloadLen, 0);

if (sent < 0) {

perror("send payload");

return -1;

}

}

return 0;

}

/\*

\* recvSMB2Response: recv an SMB2 header + payload

\*/

int recvSMB2Response(SMB2Header \*outHdr, void \*outBuf, size\_t bufSize, ssize\_t \*outPayloadLen) {

ssize\_t recvd = recv(gSock, outHdr, sizeof(SMB2Header), 0);

if (recvd <= 0) {

perror("recv SMB2 header");

return -1;

}

if (recvd < (ssize\_t)sizeof(SMB2Header)) {

fprintf(stderr, "Incomplete SMB2 header.\n");

return -1;

}

// Validate signature

if (!(outHdr->ProtocolId[0] == 0xFE &&

outHdr->ProtocolId[1] == 'S' &&

outHdr->ProtocolId[2] == 'M' &&

outHdr->ProtocolId[3] == 'B')) {

fprintf(stderr, "Invalid SMB2 signature.\n");

return -1;

}

// Non-blocking peek to see if there's more data

int peekLen = recv(gSock, outBuf, bufSize, MSG\_DONTWAIT);

if (peekLen > 0) {

int realLen = recv(gSock, outBuf, peekLen, 0);

if (realLen < 0) {

perror("recv payload");

return -1;

}

\*outPayloadLen = realLen;

} else {

\*outPayloadLen = 0;

}

return 0;

}

/\*

\* buildSMB2Header: fill out common fields

\*/

void buildSMB2Header(uint16\_t command, uint32\_t treeId, uint64\_t sessionId, SMB2Header \*hdrOut) {

memset(hdrOut, 0, sizeof(SMB2Header));

hdrOut->ProtocolId[0] = 0xFE;

hdrOut->ProtocolId[1] = 'S';

hdrOut->ProtocolId[2] = 'M';

hdrOut->ProtocolId[3] = 'B';

hdrOut->StructureSize = 64;

hdrOut->Command = command;

hdrOut->Credits = 1; // minimal

hdrOut->MessageId = gMessageId++;

hdrOut->TreeId = treeId;

hdrOut->SessionId = sessionId;

}

//--------------------------------------------------

**// SMB2 NEGOTIATE**

//--------------------------------------------------

int doNegotiate() {

SMB2Header hdr;

buildSMB2Header(SMB2\_NEGOTIATE, 0, 0, &hdr);

SMB2NegotiateRequest req;

memset(&req, 0, sizeof(req));

req.StructureSize = 36;

req.DialectCount = 3;

uint16\_t dialects[3] = {

**SMB2\_DIALECT\_0202,**

**SMB2\_DIALECT\_0210,**

**SMB2\_DIALECT\_0300**

};

// Send header + negotiate request

if (sendSMB2Request(&hdr, &req, sizeof(req)) < 0) return -1;

// Followed by the dialect array

if (send(gSock, dialects, sizeof(dialects), 0) < 0) {

perror("send dialects");

return -1;

}

// Receive

SMB2Header respHdr;

unsigned char buf[1024];

ssize\_t payloadLen;

if (recvSMB2Response(&respHdr, buf, sizeof(buf), &payloadLen) < 0) return -1;

if (respHdr.Status != STATUS\_SUCCESS) {

fprintf(stderr, "Negotiate failed, status=0x%08X\n", respHdr.Status);

return -1;

}

printf("[Client] SMB2 NEGOTIATE OK. payloadLen=%zd\n", payloadLen);

return 0;

}

//--------------------------------------------------

// SMB2 SESSION\_SETUP (stub - no real authentication)

//--------------------------------------------------

int doSessionSetup() {

SMB2Header hdr;

buildSMB2Header(SMB2\_SESSION\_SETUP, 0, 0, &hdr);

SMB2SessionSetupRequest ssreq;

memset(&ssreq, 0, sizeof(ssreq));

ssreq.StructureSize = 25;

// In real usage, you'd set SecurityBufferOffset/Length and

// provide an NTLM/Kerberos token. This is omitted here.

if (sendSMB2Request(&hdr, &ssreq, sizeof(ssreq)) < 0) return -1;

SMB2Header respHdr;

unsigned char buf[1024];

ssize\_t payloadLen;

if (recvSMB2Response(&respHdr, buf, sizeof(buf), &payloadLen) < 0) return -1;

if (respHdr.Status != STATUS\_SUCCESS) {

fprintf(stderr, "SessionSetup failed, status=0x%08X\n", respHdr.Status);

return -1;

}

gSessionId = respHdr.SessionId;

printf("[Client] SMB2 SESSION\_SETUP OK. SessionId=0x%llx\n",

(unsigned long long)gSessionId);

return 0;

}

//--------------------------------------------------

// SMB2 TREE\_CONNECT to \\server\IPC$

//--------------------------------------------------

int doTreeConnect(const char \*ipcPath) {

SMB2Header hdr;

buildSMB2Header(SMB2\_TREE\_CONNECT, 0, gSessionId, &hdr);

SMB2TreeConnectRequest tcreq;

memset(&tcreq, 0, sizeof(tcreq));

tcreq.StructureSize = 9;

tcreq.PathOffset = sizeof(tcreq);

uint32\_t pathLen = (uint32\_t)strlen(ipcPath);

tcreq.PathLength = pathLen;

size\_t reqSize = sizeof(tcreq) + pathLen;

char \*reqBuf = (char \*)malloc(reqSize);

if (!reqBuf) {

fprintf(stderr, "malloc failed\n");

return -1;

}

memcpy(reqBuf, &tcreq, sizeof(tcreq));

memcpy(reqBuf + sizeof(tcreq), ipcPath, pathLen);

if (sendSMB2Request(&hdr, reqBuf, reqSize) < 0) {

free(reqBuf);

return -1;

}

free(reqBuf);

SMB2Header respHdr;

unsigned char buf[1024];

ssize\_t payloadLen;

if (recvSMB2Response(&respHdr, buf, sizeof(buf), &payloadLen) < 0) {

return -1;

}

if (respHdr.Status != STATUS\_SUCCESS) {

fprintf(stderr, "TreeConnect to %s failed, status=0x%08X\n",

ipcPath, respHdr.Status);

return -1;

}

if (payloadLen < (ssize\_t)sizeof(SMB2TreeConnectResponse)) {

fprintf(stderr, "TreeConnect response too small\n");

return -1;

}

gTreeId = respHdr.TreeId;

printf("[Client] TREE\_CONNECT to %s OK. TreeId=0x%08X\n", ipcPath, gTreeId);

return 0;

}

//--------------------------------------------------

// SMB2 CREATE (Open named pipe, e.g. "\\PIPE\\svcctl")

//--------------------------------------------------

int doOpenPipe(const char \*pipeName) {

SMB2Header hdr;

buildSMB2Header(SMB2\_CREATE, gTreeId, gSessionId, &hdr);

SMB2CreateRequest creq;

memset(&creq, 0, sizeof(creq));

creq.StructureSize = 57;

creq.RequestedOplockLevel = 0; // none

creq.ImpersonationLevel = 2; // SecurityImpersonation

creq.DesiredAccess = 0x001F01FF; // GENERIC\_ALL (over-simplified)

creq.ShareAccess = 3; // read/write share

creq.CreateDisposition = 1; // FILE\_OPEN

creq.CreateOptions = 0;

creq.NameOffset = sizeof(SMB2CreateRequest);

// Convert ASCII to a simple UTF-16LE

uint32\_t pipeNameLenBytes = (uint32\_t)(strlen(pipeName) \* 2);

creq.NameLength = (uint16\_t)pipeNameLenBytes;

size\_t totalSize = sizeof(creq) + pipeNameLenBytes;

unsigned char \*reqBuf = (unsigned char \*)malloc(totalSize);

if (!reqBuf) {

fprintf(stderr, "malloc doOpenPipe failed\n");

return -1;

}

memcpy(reqBuf, &creq, sizeof(creq));

**// ASCII -> UTF-16LE**

unsigned char \*pName = reqBuf + sizeof(creq);

for (size\_t i = 0; i < strlen(pipeName); i++) {

pName[i\*2] = (unsigned char)pipeName[i];

pName[i\*2+1] = 0x00;

}

if (sendSMB2Request(&hdr, reqBuf, totalSize) < 0) {

free(reqBuf);

return -1;

}

free(reqBuf);

SMB2Header respHdr;

unsigned char buf[1024];

ssize\_t payloadLen;

if (recvSMB2Response(&respHdr, buf, sizeof(buf), &payloadLen) < 0) return -1;

if (respHdr.Status != STATUS\_SUCCESS) {

fprintf(stderr, "OpenPipe '%s' failed, status=0x%08X\n",

pipeName, respHdr.Status);

return -1;

}

if (payloadLen < (ssize\_t)sizeof(SMB2CreateResponse)) {

fprintf(stderr, "CreateResponse too small.\n");

return -1;

}

SMB2CreateResponse \*cres = (SMB2CreateResponse \*)buf;

gPipeFidPersistent = cres->FileIdPersistent;

gPipeFidVolatile = cres->FileIdVolatile;

printf("[Client] Named pipe '%s' opened OK. FID=(%llx:%llx)\n",

pipeName,

(unsigned long long)gPipeFidPersistent,

(unsigned long long)gPipeFidVolatile);

return 0;

}

//--------------------------------------------------

// doWritePipe: Send raw bytes into the named pipe

//--------------------------------------------------

int doWritePipe(const unsigned char \*data, size\_t dataLen) {

SMB2Header hdr;

buildSMB2Header(SMB2\_WRITE, gTreeId, gSessionId, &hdr);

SMB2WriteRequest wreq;

memset(&wreq, 0, sizeof(wreq));

wreq.StructureSize = 49;

wreq.DataOffset = sizeof(SMB2WriteRequest);

wreq.Length = (uint32\_t)dataLen;

wreq.FileIdPersistent = gPipeFidPersistent;

wreq.FileIdVolatile = gPipeFidVolatile;

size\_t totalSize = sizeof(wreq) + dataLen;

unsigned char \*reqBuf = (unsigned char\*)malloc(totalSize);

if (!reqBuf) {

fprintf(stderr, "malloc doWritePipe failed\n");

return -1;

}

memcpy(reqBuf, &wreq, sizeof(wreq));

memcpy(reqBuf + sizeof(wreq), data, dataLen);

if (sendSMB2Request(&hdr, reqBuf, totalSize) < 0) {

free(reqBuf);

return -1;

}

free(reqBuf);

// read response

SMB2Header respHdr;

unsigned char buf[512];

ssize\_t payloadLen;

if (recvSMB2Response(&respHdr, buf, sizeof(buf), &payloadLen) < 0) return -1;

if (respHdr.Status != STATUS\_SUCCESS) {

fprintf(stderr, "WritePipe failed, status=0x%08X\n", respHdr.Status);

return -1;

}

if (payloadLen < (ssize\_t)sizeof(SMB2WriteResponse)) {

fprintf(stderr, "WriteResponse too small\n");

return -1;

}

SMB2WriteResponse \*wres = (SMB2WriteResponse \*)buf;

printf("[Client] Wrote %u bytes to pipe.\n", wres->Count);

return 0;

}

//--------------------------------------------------

// doReadPipe: read back from the pipe

//--------------------------------------------------

int doReadPipe(unsigned char \*outBuf, size\_t outBufSize, uint32\_t \*outBytesRead) {

SMB2Header hdr;

buildSMB2Header(SMB2\_READ, gTreeId, gSessionId, &hdr);

SMB2ReadRequest rreq;

memset(&rreq, 0, sizeof(rreq));

rreq.StructureSize = 49;

rreq.Length = (uint32\_t)outBufSize;

rreq.FileIdPersistent = gPipeFidPersistent;

rreq.FileIdVolatile = gPipeFidVolatile;

if (sendSMB2Request(&hdr, &rreq, sizeof(rreq)) < 0) return -1;

SMB2Header respHdr;

unsigned char buf[2048];

ssize\_t payloadLen;

if (recvSMB2Response(&respHdr, buf, sizeof(buf), &payloadLen) < 0) return -1;

if (respHdr.Status != STATUS\_SUCCESS) {

fprintf(stderr, "ReadPipe failed, status=0x%08X\n", respHdr.Status);

return -1;

}

if (payloadLen < (ssize\_t)sizeof(SMB2ReadResponse)) {

fprintf(stderr, "ReadResponse too small\n");

return -1;

}

SMB2ReadResponse \*rres = (SMB2ReadResponse \*)buf;

uint32\_t dataLen = rres->DataLength;

if (dataLen > 0) {

uint8\_t \*dataStart = buf + rres->DataOffset;

// Check for bounds

if (rres->DataOffset + dataLen <= (uint32\_t)payloadLen) {

if (dataLen > outBufSize) {

dataLen = (uint32\_t)outBufSize; // Truncate

}

memcpy(outBuf, dataStart, dataLen);

} else {

fprintf(stderr, "Data offset/length out of payload bounds!\n");

return -1;

}

}

\*outBytesRead = dataLen;

printf("[Client] Read %u bytes from pipe.\n", dataLen);

return 0;

}

//--------------------------------------------------

// doDCERPCBind: a partial DCERPC bind request to SVCCTL

//--------------------------------------------------

int doDCERPCBind() {

// A typical DCERPC bind to SVCCTL might include:

// - Version/PacketType

// - Interface UUID

// - Transfer syntax, etc.

// This is an oversimplified placeholder.

unsigned char dcerpcBindStub[] = {

0x05, 0x00, // RPC version

0x0B, // bind PDU type

0x10, // flags (little-endian)

0x00, 0x00, 0x00, 0x00, // DCE call ID (placeholder)

// [Interface UUID + version], [transfer syntax], etc...

// This is incomplete for a real DCERPC bind!

};

printf("[Client] Sending partial DCERPC bind stub...\n");

return doWritePipe(dcerpcBindStub, sizeof(dcerpcBindStub));

}

//--------------------------------------------------

// doClosePipe: SMB2 Close for the named pipe handle

//--------------------------------------------------

int doClosePipe() {

SMB2Header hdr;

buildSMB2Header(SMB2\_CLOSE, gTreeId, gSessionId, &hdr);

SMB2CloseRequest creq;

memset(&creq, 0, sizeof(creq));

creq.StructureSize = 24;

creq.Flags = 0; // 0 or 1 for POSTQUERY\_ATTR

creq.FileIdPersistent = gPipeFidPersistent;

creq.FileIdVolatile = gPipeFidVolatile;

if (sendSMB2Request(&hdr, &creq, sizeof(creq)) < 0) return -1;

SMB2Header respHdr;

unsigned char buf[512];

ssize\_t payloadLen;

if (recvSMB2Response(&respHdr, buf, sizeof(buf), &payloadLen) < 0) {

return -1;

}

if (respHdr.Status != STATUS\_SUCCESS) {

fprintf(stderr, "ClosePipe failed, status=0x%08X\n", respHdr.Status);

return -1;

}

printf("[Client] SMB2 Close on pipe handle OK.\n");

return 0;

}

//--------------------------------------------------

// main()

//--------------------------------------------------

int main(int argc, char \*argv[]) {

if (argc < 3) {

fprintf(stderr, "Usage: %s <server\_ip> <server\_port>\n", argv[0]);

fprintf(stderr, "Example: %s 192.168.1.10 445\n", argv[0]);

return EXIT\_FAILURE;

}

const char \*serverIp = argv[1];

int port = atoi(argv[2]);

// 1. Create socket

gSock = socket(AF\_INET, SOCK\_STREAM, 0);

if (gSock < 0) {

perror("socket");

return EXIT\_FAILURE;

}

// 2. Connect

struct sockaddr\_in serverAddr;

memset(&serverAddr, 0, sizeof(serverAddr));

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(port);

if (inet\_pton(AF\_INET, serverIp, &serverAddr.sin\_addr) <= 0) {

perror("inet\_pton");

close(gSock);

return EXIT\_FAILURE;

}

if (connect(gSock, (struct sockaddr\*)&serverAddr, sizeof(serverAddr)) < 0) {

perror("connect");

close(gSock);

return EXIT\_FAILURE;

}

printf("[Client] Connected to %s:%d\n", serverIp, port);

**// 3. SMB2 NEGOTIATE**

if (doNegotiate() < 0) {

close(gSock);

return EXIT\_FAILURE;

}

// 4. SMB2 SESSION\_SETUP (stub)

if (doSessionSetup() < 0) {

close(gSock);

return EXIT\_FAILURE;

}

// 5. SMB2 TREE\_CONNECT to IPC$

// Construct a UNC path like "\\\\192.168.1.10\\IPC$"

char ipcPath[256];

snprintf(ipcPath, sizeof(ipcPath), "\\\\%s\\IPC$", serverIp);

if (doTreeConnect(ipcPath) < 0) {

close(gSock);

return EXIT\_FAILURE;

}

// 6. SMB2 CREATE for named pipe "\\PIPE\\svcctl"

if (doOpenPipe("\\PIPE\\svcctl") < 0) {

close(gSock);

return EXIT\_FAILURE;

}

// 7. (Optional) Send a partial DCERPC Bind

if (doDCERPCBind() < 0) {

// Not strictly fatal; you might decide to continue or bail out

fprintf(stderr, "DCERPC bind stub failed.\n");

}

// 8. Attempt a read from the pipe (whatever the server might send back)

unsigned char readBuf[512];

memset(readBuf, 0, sizeof(readBuf));

uint32\_t bytesRead = 0;

if (doReadPipe(readBuf, sizeof(readBuf), &bytesRead) < 0) {

fprintf(stderr, "Read from pipe failed.\n");

} else {

if (bytesRead > 0) {

printf("[Client] Pipe response (hex):\n");

for (uint32\_t i = 0; i < bytesRead; i++) {

printf("%02X ", readBuf[i]);

}

printf("\n");

} else {

printf("[Client] No data returned from pipe.\n");

}

}

// 9. Close the pipe handle

if (doClosePipe() < 0) {

fprintf(stderr, "Failed to close pipe properly.\n");

}

// 10. Done

close(gSock);

printf("[Client] Done.\n");

return EXIT\_SUCCESS;

}

**EXHIBIT 3: The “Eternal” family of zero-day exploits developed by the NSA, on the SMBv1 protocol**

## A Bit More Detail

**1 \*\*The Vulnerability (MS17-010)\*\***

- EternalBlue exploited a memory corruption bug in Microsoft’s SMBv1 server (in functions like `Srv!SrvOs2FeaListToNt` or `Srv!SrvTransaction2Dispatch`).

- By sending specially crafted “trans2” (transaction) packets, the attacker could write arbitrary data past buffer boundaries in kernel space (in particular, in the `SRV` driver).

**2 \*\*Named Pipe vs. Trans2\*\***

- \*\*Named Pipe Exploits (e.g., EternalRomance):\*\* Some SMB exploits from the same leak abused a named pipe—often `\pipe\SRVSVC`—to hold open a file/pipe handle in the SMB server and then manipulate buffer offsets for code execution.

- \*\*EternalBlue’s Approach:\*\* EternalBlue directly abused an out-of-bounds write in the SMBv1 “trans2” sub-protocol. While SMBv1 does support named pipes, EternalBlue’s trigger was not contingent on obtaining a pipe handle.

**3 \*\*Why the Confusion?\*\***

- All these exploits came from the same toolset (Equation Group’s FuzzBunch) and target SMB on various Windows versions.

- EternalBlue, EternalRomance, EternalChampion, and EternalSynergy each had different code paths and slightly different vulnerabilities, even though they were all SMB-related.

---

### Summary

- \*\*EternalBlue\*\* = Exploits a buffer overflow in SMBv1’s “trans2” commands.

- \*\*Does it use a pipe?\*\* No—unlike some sibling exploits (e.g., EternalRomance), it does \*\*not\*\* hinge on a named pipe handle.