Hardening Attack Surfaces with Formally Proven Message Parsers

Nikhil Swamy
Irina Spiridonova
Juan Vazquez

Tahina Ramananandro
Haobin Ni
Michael Tang

Aseem Rastogi
Dmitry Malloy
Omar Cardona

Arti Gupta

Microsoft

everest
Secure Parsing is Critical

- Improper input validation = MITRE 2020 Top #3, 2021 Top #4 most dangerous CVE software weakness

- Still a thing today in widely-used >30-year-old formats
  - Linux TCP parsing bug fix as late as 2019
  - Windows 10 Bad Neighbor (ICMPv6, 2020)

A remote code execution vulnerability exists when the Windows TCP/IP stack improperly handles ICMPv6 Router Advertisement packets. An attacker who successfully exploited this vulnerability could gain the ability to execute code on the target server or client.

To exploit this vulnerability, an attacker would have to send specially crafted ICMPv6 Router Advertisement packets to a remote Windows computer.

The update addresses the vulnerability by correcting how the Windows TCP/IP stack handles ICMPv6 Router Advertisement packets.
Handwritten parsing still around

- Handwritten C/C++ code
  - Performance, deployability (e.g. OS kernel), legacy

- Bratus et al. (Usenix Mag. 2017), LangSec:
  - “Roll your own crypto” considered harmful
  - “Roll your own parsers” also should be

- Ongoing push for automatically generated parsers
  - ProtocolBuffers, FlatBuffers, Cap’n Proto, JSON...
  - But those libraries choose the data formats
  - What about formats dictated by external constraints? (TCP, ICMP...?)
Our Solution: EverParse3D

1. Author spec

External source of truth (RFC, etc.)

Distill

Format.3d
Our Solution: EverParse3D

1. Author spec
   - External source of truth (RFC, etc.)
   - Distill
   - Format.3d

2. Proof-checking & codegen
   - EverParse3D Libs
   - F* code and proofs
     - Theorems
       - Memory safe
       - Arithmetically safe
       - Functionally correct
       - Double-fetch free
   - Format.fst
   - Automatically translate
   - Auto. verify & code gen
   - Format.c
Our Solution: EverParse3D

1. Author spec
   - External source of truth (RFC, etc.)
   - Distill
   - Format.3d

2. Proof-checking & codegen
   - EverParse3D Libs
   - Theorems
     - Memory safe
     - Arithmetically safe
     - Functionally correct
     - Double-fetch free
   - F* code and proofs
   - Auto. verify & code gen
   - Format.fst

C/C++ application
   - Handwritten parser
   - Format.c

1. Automate translation
2. Proof checking & codegen

C/C++ application
Our Solution: EverParse3D

1. Author spec
External source of truth (RFC, etc.)

Distill

Format.3d

2. Proof-checking & codegen
F* code and proofs

EverParse3D Libs

Theorems
- Memory safe
- Arithmetically safe
- Functionally correct
- Double-fetch free

Format.fst

Automatically translate

3. Integrate
C/C++ application

Auto. verify & code gen

Format.c

1. Author spec

2. Proof-checking & codegen

3. Integrate
Our Solution: EverParse3D

1. Author spec
   - External source of truth (RFC, etc.)
   - Distill
   - Format.3d

2. Proof-checking & codegen
   - EverParse3D Libs
   - Format.3d
   - Automatically translate
   - Format.fst
   - F* code and proofs
   - Theorems
     - Memory safe
     - Arithmetically safe
     - Functionally correct
     - Double-fetch free
   - Auto. verify & code gen
   - Format.c

3. Integrate
   - C/C++ application
   - Now in Windows and Azure network virtualization
EverParse3D Guarantees

- Memory safety: no buffer overrun
- Arithmetic safety: no integer overflow

```c
uint32_t fld_offset = input[current];
uint32_t fld = input[current+offset];
```

Missing checks for integer/buffer overflows
EverParse3D Guarantees

- Memory safety: no buffer overrun
- Arithmetic safety: no integer overflow

- Functional correctness:
  - All ill-formed packets are rejected
  - Every valid packet is accepted

```c
uint32_t fld_offset = input[current];
uint32_t fld = input[current+offset];
```

Missing checks for integer/buffer overflows
EverParse3D Guarantees

- Memory safety: no buffer overrun
- Arithmetic safety: no integer overflow

- Functional correctness:
  - All ill-formed packets are rejected
  - Every valid packet is accepted

- Double-fetch freedom: no “time-of-check to time-of-use” bugs
  - No exclusive read access to the input buffer
  - Solution: Read each byte at most once
  - Validation on a “logical snapshot” of the input data

```c
uint32_t fld_offset = input[current];
uint32_t fld = input[current+offset];
```

Missing checks for integer/buffer overflows
typedef struct _TCP_HEADER
{
    ...
    UINT16 CWR:1;  UINT16 ECE:1;  UINT16 URG:1;  UINT16 ACK:1;
    UINT16 PSH:1;  UINT16 RST:1;  UINT16 SYN:1;  UINT16 FIN:1;  ...
    URGENT_PTR UrgentPointer;
    OPTION          Options   [];
    UINT8           Data      [];
} TCP_HEADER;

typedef union _OPTION_PAYLOAD {
    all_zeros EndOfList;
    unit Noop;
    ...
} OPTION_PAYLOAD;

typedef struct _OPTION {
    UINT8 OptionKind;
    OPTION_PAYLOAD OptionPayload;
} OPTION;
Augmenting C data types with value constraints,

typedef union _OPTION_PAYLOAD {
    UINT8          all_zeros;
    UINT8          EndOfList;
    UINT8          unit;
    UINT8          Noop;
    ...}

typedef struct _OPTION {
    UINT8          OptionKind;
    _OPTION_PAYLOAD OptionPayload;
} OPTION;

typedef struct _TCP_HEADER {
    ...
    UINT16 CWR:1;  UINT16 ECE:1;  UINT16 URG:1;  UINT16 ACK:1;
    UINT16 PSH:1;  UINT16 RST:1;  UINT16 SYN:1;  UINT16 FIN:1;  ...
    URGENT_PTR UrgentPointer {UrgentPointer == 0 || URG == 1 } ;
    OPTION          Options   [];
    UINT8           Data      [];
} TCP_HEADER;
3D: A source language of message formats for Dependent Data Descriptions

Augmenting C data types with value constraints, variable-length structures

typedef struct _TCP_HEADER(UINT32 SegmentLength)
{
...
UINT16 CWR:1; UINT16 ECE:1; UINT16 URG:1; UINT16 ACK:1;
UINT16 PSH:1; UINT16 RST:1; UINT16 SYN:1; UINT16 FIN:1; ...
URGENT_PTR UrgentPointer {UrgentPointer == 0 || URG == 1 } ;

OPTION Options [:byte-size (DataOffset * 4) - sizeof(this)];
UINT8 Data [SegmentLength - (DataOffset * 4)];
} TCP_HEADER;

typedef union _OPTION_PAYLOAD {
all_zeros EndOfList;

unit Noop;
...
} OPTION_PAYLOAD;

typedef struct _OPTION {
UINT8 OptionKind;
OPTION_PAYLOAD OptionPayload;
} OPTION;
3D: A source language of message formats for Dependent Data Descriptions

Augmenting C data types with value constraints, variable-length structures, value-dependent unions

typedef struct _TCP_HEADER(UINT32 SegmentLength)
{
    ...
    UINT16 CWR:1;  UINT16 ECE:1;  UINT16 URG:1;  UINT16 ACK:1;
    UINT16 PSH:1;  UINT16 RST:1;  UINT16 SYN:1;  UINT16 FIN:1;  ...
    URGENT_PTR UrgentPointer {UrgentPointer == 0 || URG == 1 } ;

    OPTION(SYN==1) Options [:byte-size (DataOffset * 4) - sizeof(this)];
    UINT8 Data [SegmentLength - (DataOffset * 4)];
} TCP_HEADER;

castype _OPTION_PAYLOAD
( UINT8 OptionKind, Bool MaxSegSizeAllowed ) {
    switch(OptionKind) {
        case OPTION_KIND_END_OF_OPTION_LIST:
            all_zeros EndOfList;
        case OPTION_KIND_NO_OPERATION:
            unit Noop;
        ...
    }
} OPTION_PAYLOAD;

typedef struct _OPTION(Bool MaxSegSize) {
    UINT8 OptionKind;
    OPTION_PAYLOAD(OptionKind, MaxSegSize)
    OptionPayload;
} OPTION;
3D: A source language of message formats for Dependent Data Descriptions

Augmenting C data types with value constraints, variable-length structures, value-dependent unions and actions

typedef struct _TCP_HEADER(UINT32 SegmentLength, mutable URGENT_PTR *Dst) {
    ...
    UINT16 CWR:1;  UINT16 ECE:1;  UINT16 URG:1;  UINT16 ACK:1;
    UINT16 PSH:1;  UINT16 RST:1;  UINT16 SYN:1;  UINT16 FIN:1;  ...
    URGENT_PTR UrgentPointer {UrgentPointer == 0 || URG == 1 }
    {:on-success *Dst = UrgentPointer; }

    OPTION(SYN==1) Options [byte-size (DataOffset * 4) - sizeof(this)];
    UINT8 Data [SegmentLength - (DataOffset * 4)];
} TCP_HEADER;

casetype _OPTION_PAYLOAD (UINT8 OptionKind, Bool MaxSegSizeAllowed) {
    switch(OptionKind) {
        case OPTION_KIND_END_OF_OPTION_LIST: all_zeros EndOfList;
        case OPTION_KIND_NO_OPERATION: unit Noop;
        ...
    }
} OPTION_PAYLOAD;

typedef struct _OPTION(Bool MaxSegSize) {
    UINT8 OptionKind;
    OPTION_PAYLOAD(OptionKind, MaxSegSize)
    OptionPayload;
} OPTION;
3D: A source language of message formats for Dependent Data Descriptions

Also in the paper:
- Full formalization of the language in F*
- 3 denotational semantics of a hybrid shallow-deep embedding
- Built on top of dependently-typed monadic parsing combinators (USENIX 2019)
- Via partial evaluation and 1st Futamura projection
- Yields high-performance C code via Karamel F*-to-C compiler (ICFP 2017)

```c
typedef struct _TCP_HEADER(UINT32 SegmentLength, mutable URGENT_PTR *Dst) {
    ...
    UINT16 CWR:1;  UINT16 ECE:1;  UINT16 URG:1;  UINT16 ACK:1;
    UINT16 PSH:1;  UINT16 RST:1;  UINT16 SYN:1;  UINT16 FIN:1;  ...
    URGENT_PTR UrgentPointer
        { UrgentPointer == 0 || URG == 1 };
    OPTION(SYN==1) Options[:byte-size(DataOffset*4) - sizeof(this)];
    UINT8 Data[SegmentLength - (DataOffset * 4)];
} TCP_HEADER;

typedef struct _OPTION(Bool MaxSegSize) {
    UINT8 OptionKind;
    OPTION_PAYLOAD(OptionKind, MaxSegSize) OptionPayload;
} OPTION;
```
This paper: Verified Secure Parsers for Microsoft Hyper-V Network Virtualization

- Hyper-V: Hypervisor for Windows 10, 11, and all Azure Cloud

- vSwitch: Dispatches network packets from/to guests
This paper: Verified Secure Parsers for Microsoft Hyper-V Network Virtualization

- Hyper-V: Hypervisor for Windows 10, 11, and all Azure Cloud

- vSwitch: Dispatches network packets from/to guests

- Some guest-side optimizations to give some direct hardware access (VMBUS), bypassing a hypercall
This paper: Verified Secure Parsers for Microsoft Hyper-V Network Virtualization

- Hyper-V: Hypervisor for Windows 10, 11, and all Azure Cloud

- vSwitch: Dispatches network packets from/to guests

- Some guest-side optimizations to give some direct hardware access (VMBUS), bypassing a hypercall

- Need to protect against attacks from network or malicious guests crafting ill-formed packets to break isolation / gain host access
Hyper-V vSwitch: network packet layers

Talk to the hypervisor

Talk to the network
Hyper-V vSwitch: network packet layers

Beginning of packet

Talk to the hypervisor

Talk to the device

End of packet

vSwitch

Host

NetVsc

Guest

Talk to the device
Hyper-V vSwitch with EverParse3D

- **Now in Windows 10, 11, and Azure Cloud**: Every network packet passing through Hyper-V is validated by EverParse3D formally verified code

- NVSP, RNDIS, OIDs and NDIS
  - Some of which are proprietary
  - Other formats (TCP, etc.) in progress

- 5K lines of 3D specification
  - 137 structs, 22 casetypes, 30 enum types

- Verified in 82 s
- Generated 23K C code
Performance

Generated code is fast...

- Our code passed internal performance regression testing, imposing less than 2% cycles/byte overhead
- In some cases, our code is more efficient by virtue of eliminating unneeded copies

... thanks to careful design

- Validators operate in-place
- Validators only read data at most once: client code no longer needs to copy data before validating it
- Layered specifications + one single pass = fail early

Detailed performance results contain proprietary information, thus are not included in the paper
A multi-year (since summer 2019), multi-org effort

Research Team  Product Team  Testing Team  Security Team
A multi-year, multi-org effort

Research Team       Product Team

Gather requirements:
• Parsing actions
• Double-fetch freedom
• <2% perf overhead
• Generated C code quality (guidelines, etc.)
A multi-year, multi-org effort

Figure out the data format specification:
• Some protocols have no pre-existing specs
• Backward compatibility
A multi-year, multi-org effort

Figure out the data format specification:
• Some protocols have no pre-existing specs
• Backward compatibility
• Complex testing matrices
A multi-year, multi-org effort

Security evaluation:
• Spec audited, security team wrote unit tests
• vSwitch code fuzzers stopped finding bugs:
  • Malformed packets properly rejected by our parsers
  • Helped refocus fuzzers to functionality fuzzing
A multi-year, multi-org effort

Productivity improvements:
• EverParse3D now part of the Windows build environment (incl. Z3, F*, Karamel)
• Critical to meet product deadlines:
  • saves code writing cost
  • more focused security reviews
A multi-year, multi-org effort

Active Maintenance (2 years already):
• Product teams change the specs as they integrate new features
• Backport to older product versions
• Generated C code checked in the product repo to aid other teams’ understanding

Product Team  Testing Team  Security Team  + Other teams (servicing, etc.)
EverParse3D Takeaway

• A sweet spot for formal verification
  • Strong mathematical guarantees of memory safety and functional correctness
  • Provably correct by construction: Zero user proof effort
  • High-performance code generated from data format description in a high-level declarative language
  • High return on investment wrt. attack surface

• Project page and manual: https://project-everest.github.io/everparse/
  • Open-source (Apache 2 license)
  • Binary releases for Linux and Windows