PNW PLSE
University of Washington
May 9th 2023

eVerpΛrse

Hardening critical attack surfaces with formally proven message parsers

Tahina Ramananandro (RiSE @MSR Redmond)
Nikhil Swamy (RiSE @MSR Redmond)
Aseem Rastogi (MSR Bangalore)

Microsoft

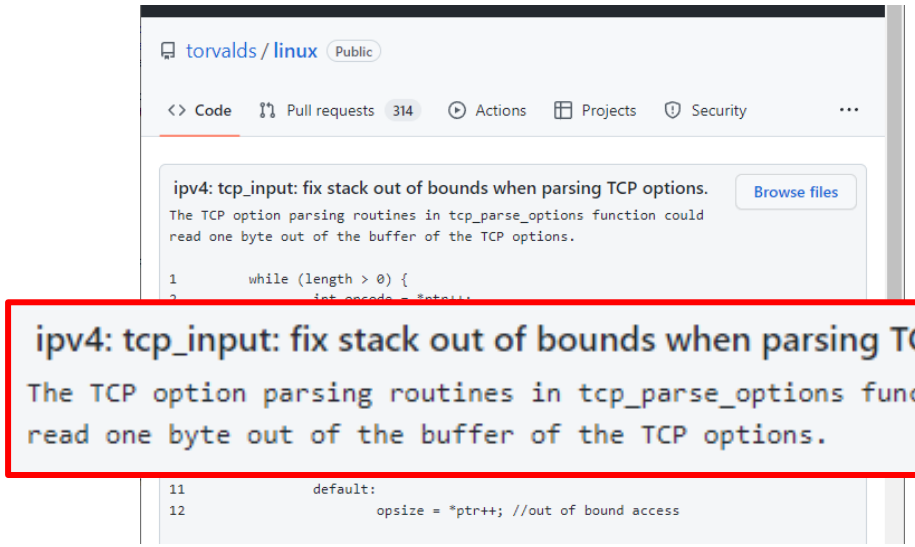
RiSE

everest
Verified End-to-End Secure Transport

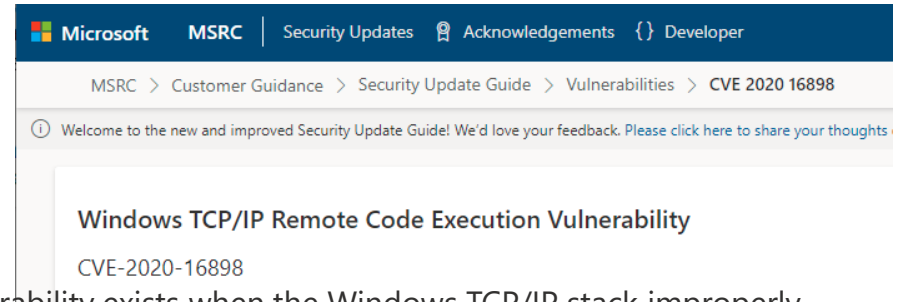
Secure Parsing is Critical

- Improper input validation = MITRE 2020 Top #3, 2021/22 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)



ipv4: tcp_input: fix stack out of bounds when parsing TCP options.
The TCP option parsing routines in tcp_parse_options function could read one byte out of the buffer of the TCP options.



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.

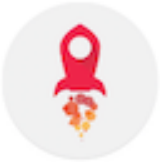
The update addresses the vulnerability by correcting how the Windows TCP/IP stack handles ICMPv6 Router Advertisement packets.

Handwritten parsing still around

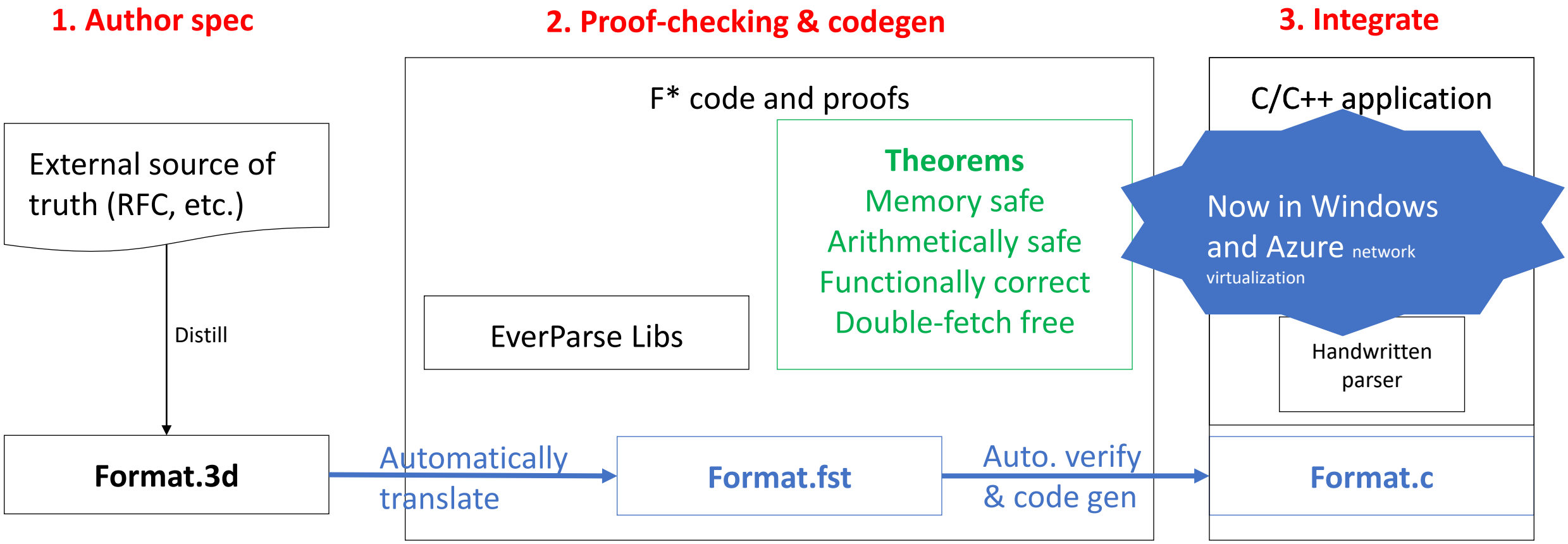
- Handwritten C/C++ code
  - Performance, deployability (e.g. OS kernel), legacy

- 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,…
  - But those libraries choose the data formats
  - **What about formats dictated by external constraints?** (TCP, ICMP…)



Our Solution: EverParse



EverParse Guarantees

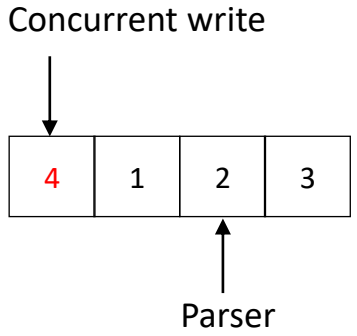
- Memory safety: no buffer overrun
- Arithmetic safety: no integer overflow

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

Missing checks for integer/buffer overflows

- 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



3D: A source language of message formats for Dependent Data Descriptions

```
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;
```

3D: A source language of message formats for Dependent Data Descriptions

Augmenting C data types with value constraints,

```
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;
```

```
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

```
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;
```

```
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

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;
```

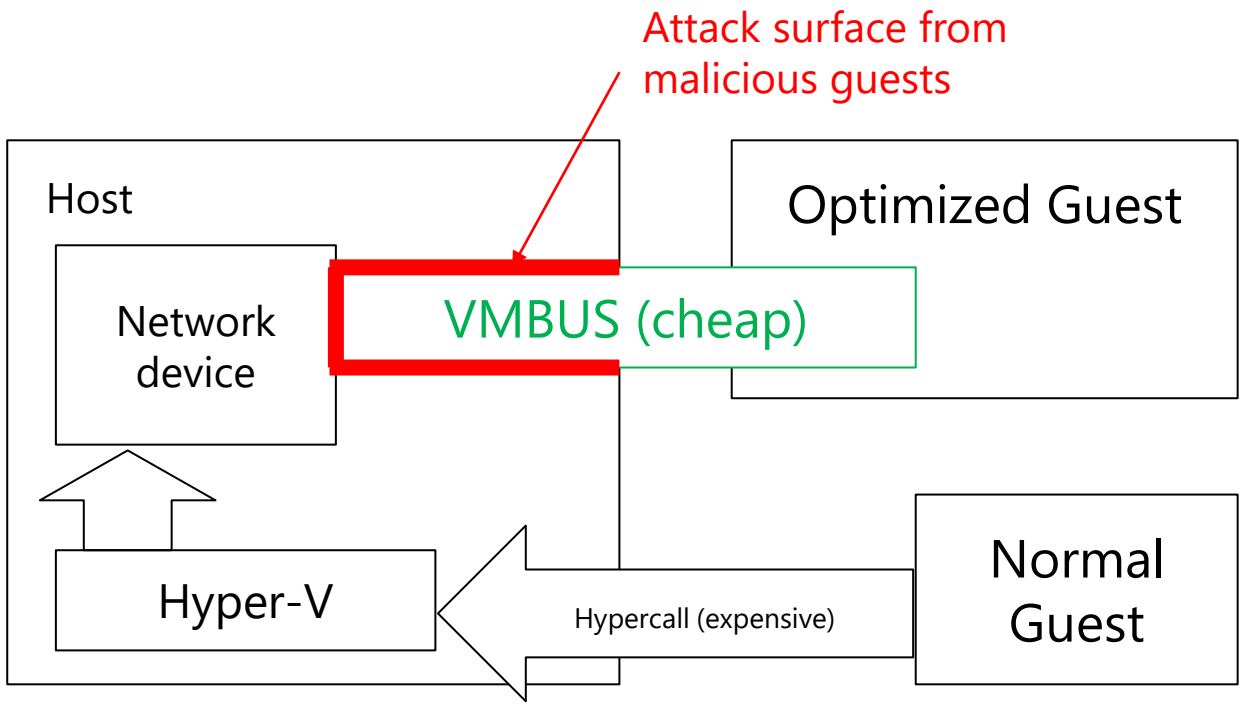
```
casetype _OPTION_PAYLOAD
  (UINT8 OptionKind, Bool MaxSegSizeAllowed) {
  switch(OptionKind)  {
     case OPTION_KIND_END_OF_OPTION_LIST:
       all_zeros EndOfList;
     case OPTION_KIND_NO_OPERATION:
       unit Noop;
     …
}} OPTION_PAYLOAD;

typedef struct _OPTION(Bool MaxSegSize) {
    UINT8 OptionKind;
    OPTION_PAYLOAD(OptionKind, MaxSegSize)
          OptionPayload;
} OPTION;
```

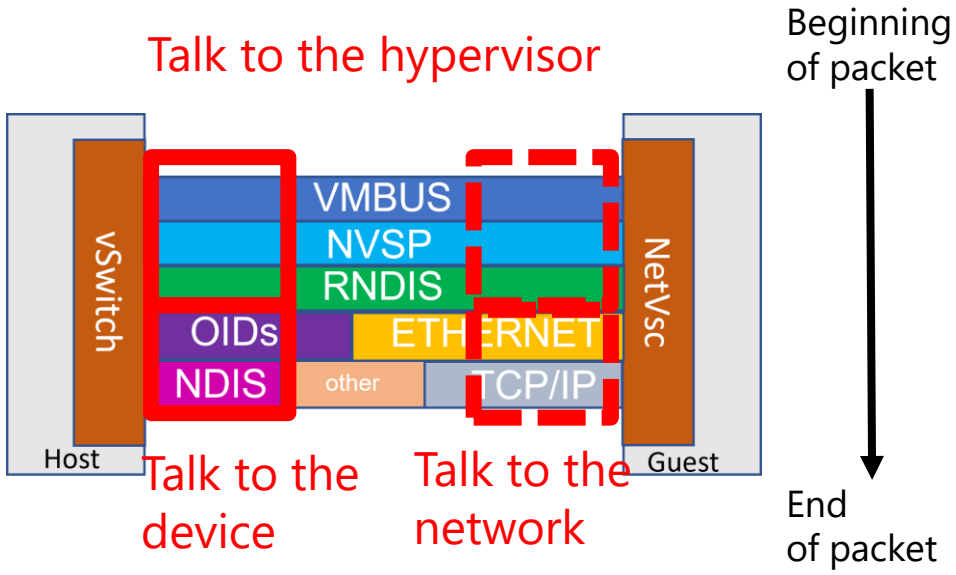
Microsoft Hyper-V vSwitch with EverParse3D (PLDI 2022)

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

5K lines of 3D specification

- Verified in 82 s
- Generated 23K C code
- Less than 2% cycles/byte overhead
- In some cases, our code is more efficient by virtue of eliminating unneeded copies





Using EverParse with Verified F* Applications

TLS

- Verified TLS secure channel with formal security model
- Handshake message formats
- Verified non-malleable (unique binary representation, for signature-based authentication)
- USENIX Security 2019

QUIC

- Verified QUIC record layer with formal security model
- Parsing and serialization proven constant-time for side-channel resistance
- IEEE S&P 2021

DICE/RIoT

- Verified measured boot for embedded devices (secured boot with measurements)
- Right-to-left serialization for length-prefixed data
- ASN.1 X.509 certificate subset
- USENIX Security 2021

EverParse Takeaway

- A sweet spot for formal verification
  - Strong guarantees of memory safety, functional correctness and security
  - Provably correct by construction: Zero user proof effort
  - Handwritten parsers are a thing of the past
  - High return on investment wrt. attack surface
  - Other formats in progress (CBOR, ASN.1,…)

- Project page and manual: https://project-everest.github.io/everparse/
  - Open-source (Apache 2 license)
  - Binary releases for Linux and Windows
- For more info: {taramana,nswamy,aseemr}@microsoft.com
  - Come to LangSec @ IEEE S&P 2023 to learn more about EverParse (I'll give a talk there too)