HTTP/2 Attacks Generation using 5Greplay

Francesco G. Caccavale, Huu Nghia Nguyen, Ana R. Cavalli,
Edgardo Montes de Oca, Wissam Mallouli
firstname.lastname@montimage.com
Montimage
Paris, France

ABSTRACT

5G networks become increasingly pervasive, ensuring the robustness and integrity of network functions. The adoption of HTTP/2 in 5G core functions brings notable performance benefits but also introduces potential security risks. By analyzing HTTP/2 related threats, this research aims to shed light on the security challenges faced by 5G networks. The paper proposes effective security testing methodologies using an open-source solution called 5Greplay to detect these security breaches, enabling network operators to protect against potential attacks, safeguard user privacy, and ensure uninterrupted service continuity. By addressing the specific concerns of HTTP/2 related threats, this research contributes to the overall security posture of 5G network functions and provides valuable insights for the secure deployment of 5G networks in an evolving threat landscape.

KEYWORDS

HTTP/2, Attack Injection, Fuzz Testing, Traffic Engineering, 5G

ACM Reference Format:
Francesco G. Caccavale, Huu Nghia Nguyen, Ana R. Cavalli,, Edgardo Montes de Oca, Wissam Mallouli . 2023. HTTP/2 Attacks Generation using 5Greplay. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29-September 1, 2023, Benevento, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3600160.3605062

1 INTRODUCTION

Security, trust and reliability are crucial issues in mobile 5G networks from both hardware and software perspectives [1]. These issues are significant when considering implementations over distributed environments, e.g., corporate Cloud environments over massively virtualized infrastructure as envisioned in the 5G service provision paradigm. The virtualized nature of the infrastructure introduces new vulnerabilities and risks. Proper isolation and segmentation of virtual resources, secure hypervisor implementations, and strong access controls are essential to prevent unauthorized access or lateral movement within the infrastructure.

Reliability is another critical aspect, especially in mission-critical applications such as autonomous vehicles, remote healthcare, and industrial automation. High availability, fault tolerance, and redundancy mechanisms should be in place to ensure uninterrupted service delivery. Load balancing and failover mechanisms can help distribute the network load and ensure smooth operations even in the face of failures or network congestion [4].

To address these security, trust, and reliability concerns, rigorous testing, auditing, and continuous monitoring are necessary. Regular security assessments, vulnerability scanning, and penetration testing should be conducted to identify and mitigate potential risks. Compliance with industry standards and regulations, such as those set forth by 3GPP (Third Generation Partnership Project) [14] and relevant security bodies, is also crucial to maintain a secure and trusted 5G network environment.

Overall, addressing security, trust, and reliability issues in mobile 5G networks is a multifaceted task that requires a holistic approach, involving both hardware and software considerations, in addition to proper governance, policies, and ongoing monitoring.

This work is part of the SANCUS project[1], whose proposed solution aims to provide a modular framework integrating different engines to enable next generation 5G system networks to perform automated operations and intelligent analysis of their large-scale firmware images, as well as validation of applications and services. SANCUS also proposes a proactive risk assessment of network applications and services by maximising the overall system resilience in terms of security, privacy and reliability, and an inclusive solution for modelling and emulating network container services and applications, along with network-wide attacks, forensic investigations, and tests that require a safe environment without the risk of proprietary data loss or adverse impact upon existing networks. The strength of the SANCUS test engines is that they allow testing not only large-scale network infrastructures, but also emulating the end-users (IoT, routers, hotspots). One of the main components of the SANCUS framework is the 5Greplay [11–13] solution designed by Montimage.

5Greplay is an open-source 5G fuzzer and attack injector that allows forwarding 5G network packets from one network interface card (NIC) to another with or without modifications. It can be considered as a one-way bridge between the input NIC and the output one. It can also take as input pre-captured 5G packets that are saved in a PCAP-format file. Its behavior is controlled by user defined rules and completed by a configuration file. The user defined rules allow explicitly indicating which packets can be passed through the bridge and how a packet is to be modified in the bridge. The configuration file allows specifying the default actions to be applied on the packets that are not managed by the rules, i.e., if they should be forwarded or not. Thanks to its ability to create a variety of 5G network traffic scenarios, 5Greplay enables the implementation of cyberattacks, such as those identified by ENISA [2], as well as the security test cases proposed by the 3GPP [14].

The paper is organized as follows. Section 2 illustrates the fundamentals of HTTP/2 protocol and the existing relations between HTTP/2 and 5G. Section 3 presents the related work. We introduce in Section 4 an extension of 5Greplay to deal with HTTP/2 protocol. Section 5 introduces the HTTP/2 related threats and attack scenarios that have been conducted to experimentally evaluate our

[1]https://www.sancus-project.eu/

implementation. We also present some discussion on the obtained results in this section. Finally, we conclude the paper in Section 6.

2 BACKGROUND

2.1 HTTP/2 protocol

The HTTP/2 protocol is the version following the HTTP/1.1 protocol. It has been designed to improve the efficiency and the throughput of the online communications, and it introduces some important updates. In particular, in order to solve the problem of the slow responses in HTTP/1.1, it introduces the concept of multiplexing of the streams.

Multiplexing is a technique that allows grouping different requests and responses within so-called streams, thus eliminating the need to re-establish new TCP connections each time. Each stream is identified by a stream ID, and the protocol allows for multiple streams in parallel. In order to handle this amount of data, HTTP/2 relies on its own flow control mechanism based on ad-hoc frames, such as WINDOW_UPDATE frame.

Another important feature that differentiates HTTP/2 from previous versions is that, despite being an application protocol, it does not use textual strings for communication, but employs binary data. The framing speeds up parsing by the server and reduces overhead but it makes data more difficult to interpret for developers. HTTP/2 improves the latency and reduces the amount of data exchanged between the clients and the servers using header compression. Since the previous compression algorithms were prone to security issues, HTTP/2 employs a new algorithm to compress the header, namely Hpack. Hpack uses a static dictionary containing the standard header fields, a dynamic dictionary containing the values and header fields received during communication, and finally Huffman encoding to encode the strings in name-value format.

In terms of security, HTTP/2 provides a secure communication through Transport Layer Security (TLS) protocol, which employs cryptography to protect data exchanged between client and server. HTTP/2 establishes the cypher suites allowed in order to avoid the selection of weak or old algorithms [10].

2.2 HTTP/2 and 5G networks

In 5G networks, several network functions use HTTP/2 in their API communications to enhance the performance and enable efficient data exchanges. Some of the key network functions that employ HTTP/2 in their API communication include:

- 5G Core Network Functions (NFs): Various network functions within the 5G core network, such as the Access and Mobility Management Function (AMF), Session Management Function (SMF), User Plane Function (UPF), and Network Slice Selection Function (NSSF), may use HTTP/2 in their API communications. These network functions often interact with each other using APIs for tasks such as session establishment, user authentication, mobility management, and network slicing management.
- Application Function (AF): The Application Function in the 5G core networks is responsible for hosting and managing applications and services, and may use HTTP/2 for API communications. This enables efficient exchange of data and commands between the AF and other network functions, allowing the AF to control and coordinate service delivery over the 5G network.
- Policy Control Function (PCF): The Policy Control Function is responsible for policy enforcement and management in the 5G core network, and may employ HTTP/2 in its API communications. The PCF uses APIs to communicate with other network functions, such as the AF and SMF, to enforce policy rules, apply quality-of-service (QoS) parameters, and manage resource allocation based on policy decisions.
- Authentication Server Function (AUSF): The Authentication Server Function in the 5G core network uses HTTP/2 for its API communications. This allows interacting with other network functions, such as the AMF and SMF, to authenticate users, authorize access to services, and provide secure identity management within the 5G network.
- Charging Function (CHF): The Charging Function in the 5G core network is responsible for tracking and managing the charging information related to the services used by subscribers. It may utilize HTTP/2 in its API communications with other network functions. By leveraging HTTP/2-based APIs, the CHF can efficiently exchange charging data with functions such as the Policy Control Function (PCF) and Application Function (AF) to ensure accurate charging and billing for the services consumed by users in the 5G network.
- Network Exposure Function (NEF): The Network Exposure Function enables external applications to interact with the 5G network and access network resources through APIs. HTTP/2 may be used in the API communications between the NEF and external applications, allowing efficient data exchange and service provisioning.
- Network Slice Management Function (NSMF): The Network Slice Management Function is responsible for the creation, orchestration, and management of network slices in 5G networks. HTTP/2 can be employed in the API communications between the NSMF and other network functions to exchange information related to slice instantiation, configuration, and monitoring.
- Application Function (APP): Besides the AF mentioned earlier, other application functions hosted within the 5G core network may also use HTTP/2 for their API communications. These application functions can leverage the benefits of HTTP/2 to enhance their interactions with other network functions and deliver optimized services over the 5G network.
- Network Data Analytics Function (NWDAF): The Network Data Analytics Function collects and analyzes network data to derive insights and enable intelligent network optimizations. HTTP/2 may be used in the API communications between the NWDAF and other network functions to exchange data and analytics results, facilitating data-driven decision-making within the 5G network.
- Unified Data Management (UDM): The Unified Data Management function is responsible for managing subscriber data in the 5G core network. HTTP/2 may be employed in the API communications between the UDM and other network functions to exchange subscriber-related information,

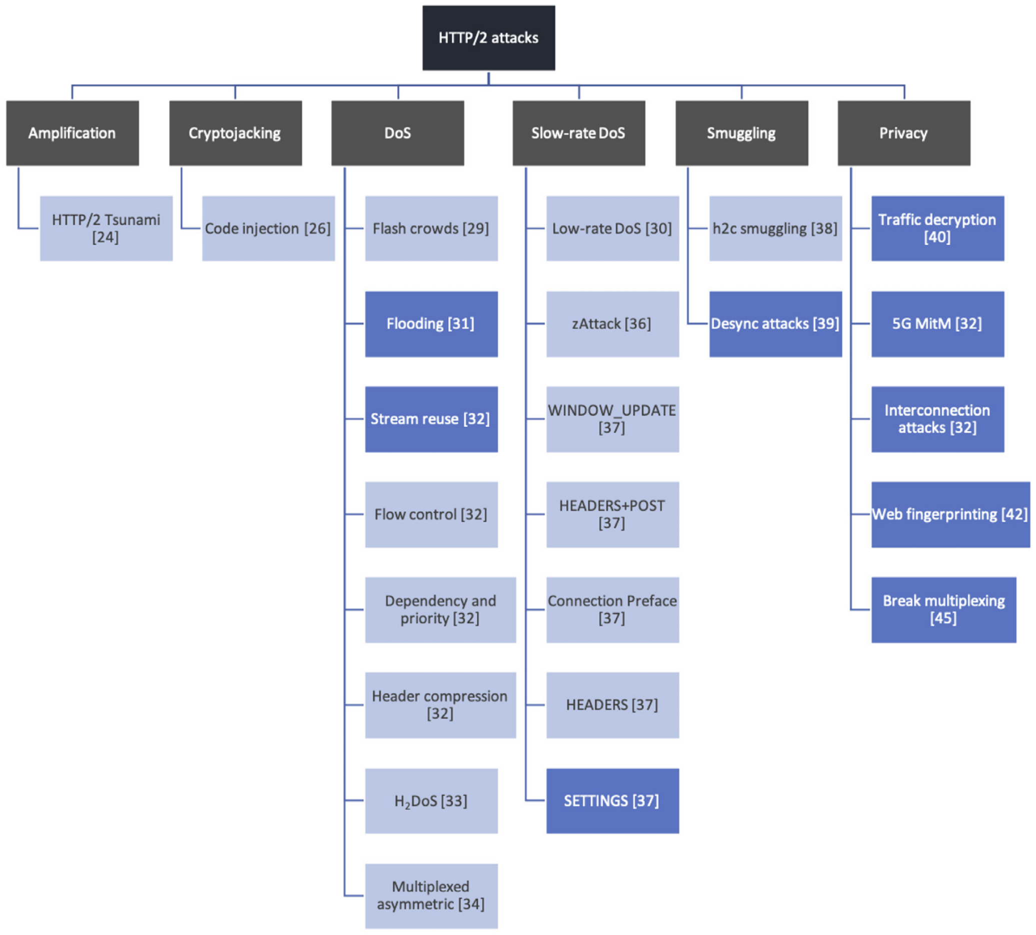


Figure 1: Taxonomy of attacks against HTTP/2 (light blue: attacks against HTTP/2 only, blue: attacks against HTTP/2 that apply also to HTTP/3) [5]

perform user authentication, and enable secure access to subscriber data.

It's important to note that the specific network functions and their implementation may vary across different 5G deployments, and the use of HTTP/2 in API communications is dependent on the design choices and requirements of the network operators and vendors involved. [16]

3 RELATED WORK

5G networks are a paradigm of next generation wireless technology. They will be faster and able to handle more connected devices than the existing 4G networks. They are composed of different enabling technologies, such as Software-Defined Networking (SDN), Network Function Virtualisation (NFV), Multi-access Edge Computing (MEC), Cloud-native Core Network (CCN) and Network Resource Slicing (NRS). Although these technologies have been the subject of various research works on cybersecurity, they introduce a new set of cybersecurity challenges that still need to be investigated. The evolution of 5G mobile networks towards a service-based architecture (SBA) comes with the emergence of numerous new testing challenges and objectives. Regarding security testing, 5G issues have been the subject of numerous studies [6–9]. Standardization organisms list collections of threats and vulnerabilities that are also investigated by academia and industrial researchers. However, there is no specific tool on the market that allows easy 5G security testing to verify if its components are protected against reported security issues.

In particular, regarding cyberattacks, different techniques have been developed capable of detecting and executing them. One of the techniques used is fuzz testing, which is a software testing technique that consists of the injection of random, invalid or unexpected data to cause the malfunctioning or a crash of the system.

The authors in [5] define a taxonomy for HTTP/2 cyberattacks presented in Figure 1 . The authors of this paper identified the main types of HTTP/2 based cyberattacks:

- Amplification attacks are a very common DDoS attack which seeks to consume the capacity of the target's network link

by reflecting low bandwidth traffic off a reflection source which provides a larger data response. The use of header compression by HTTP/2 may give an attacker a methodof amplification as the headers will have to be decompressed before processing.
- Cryptojacking is about consuming resources for mining cryptocurrencies without the consent of the resources' owner
- Denial of Service attacks
- Slow rate attacks are based on sending low-rate traffic that contains resource-hungry instructions to a victim HTTP/2 server
- Smuggling exploits the inconsistency in parsing non-RFC-compliant HTTP requests via two HTTP devices (generally a backend server and HTTP-enabled firewall or a front-end proxy). The HTTP request smuggling process is carried out by creating multiple, customized HTTP requests that make two target entities see two distinct series of requests.
- Privacy attacks

In this paper, we focus on how to use fuzz testing the generate HTTP/2 attacks targeting 5G core functions. Notice that a monitoring solution called MMT-5G[2] is also developed to detect such attacks.

4 GENERATION OF HTTP/2 ATTACKS USING 5GREPLAY

5Greplay is the Montimage tool used to replay or modify traffic towards 5G components. The main workflow of 5Greplay is depicted in Figure 2. The input of 5Greplay is network traffic, which is in the form of a PCAP file or live network data, a set of mutation rules, and a configuration file. Once a packet is processed by the tool, the context written by the user in the mutation rules will determine if the packet will be mutated or not. If the packet is to be mutated, the specified action embedded in the mutation rules will determine what type of mutation must be done in the packet. The packet is then forwarded to the output NIC, together with the non-modified packets, depending on the default action contained in the configuration file. The output packets can also be saved as a PCAP file which can be used for further investigations and repeating the tests.

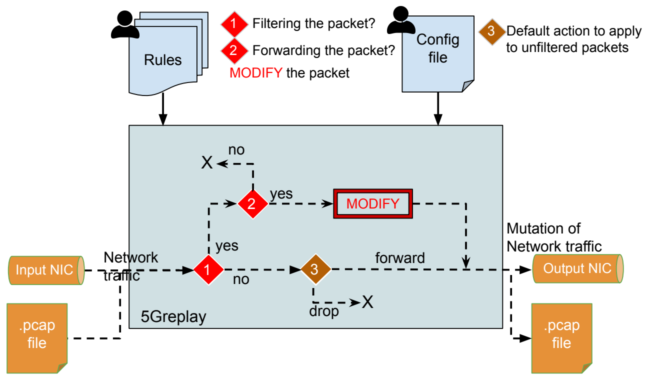


Figure 2: 5Greplay main architecture

[2]https://github.com/Montimage

5Greplay relies on MMT-DPI part of the open-source MMT security monitoring framework developed by Montimage called MMT[2]. This framework includes (among other tools) probes (MMT-Probe) to analyse structured communications, the C library that implements the Deep Packet Inspection (DPI) technique (MMT-DPI), another C library to define rules and algorithms to detect anomalies (MMT-Security), and a web application to display statistics and alarms (MMT-Operator). MMT-DPI analyses network packets in order to classify the network protocol being used or the application a packet belongs to, and extracts network and application based events, such as, protocols attribute values, network and application QoS parameters. DPI is an advanced technique [15] that classifies packets by examining the contents of packets passing through a given checkpoint and a set of signatures in the headers and payloads in order to obtain more accurate results. MMT-DPI has a plugin architecture for easily adding new protocols and analysis techniques (including machine learning techniques). It provides a set of public APIs for integration with third party probes.

Here, a plugin of MMT-DPI was developed to parse and mutate the HTTP/2 protocol packets. The plugin checks if the first characters coincide with the protocol's signature, and if so, identifies the HTTP/2 protocol at the application layer. After the identification, the plugin gets the application layer offset and extracts HTTP/2 attribute values useful for the attacks:

- HTTP/2_HEADER_LENGTH: three bytes indicating the size of the header
- HTTP/2_TYPE: one byte that defines the type of the frame. The most common types are DATA, HEADERS, PRIORITY, RST_STREAM,SETTINGS, PUSH_PROMISE, GOAWAY, WINDOW_UPDATE.
- HTTP/2_HEADER_STREAM_ID : odd number that identifies the stream of the communication in the header
- HTTP/2_PAYLOAD_LENGTH : three bytes indicating the size of the payload
- HTTP/2_PAYLOAD_STREAM_ID: odd number that identifies the stream of the communication in the payload
- HTTP/2_PAYLOAD_DATA: pointer to the beginning of the data in the payload.

Once that key attributes are extracted through MMT-DPI, 5Greplay can use them for generating the attack traffic. In particular, for DOS attacks, 5Greplay modifies for each stream the Stream ID in the header and in the payload with increasing numbers, otherwise the server will reject the request. In fuzzing attacks, instead, 5Greplay uses the HTTP/2_PAYLOAD_DATA pointer to access to the packet's payload and inject random symbols.

5 ATTACK SCENARIOS

After the integration of the HTTP/2 plugin in MMT-DPI, several attacks have been executed against an HTTP/2 server. The setup of the experiments involved the creation of the server using the javascript programming language that simulates a response for each of the possible incoming requests. The javascript language was chosen for the creation of the server because, unlike other languages, the latter provides better support and more complete documentation for the HTTP/2 protocol. Both the aforementioned server and 5Greplay were deployed inside a VirtualBox virtual machine running on the Ubuntu version 22.04 operating system. In order to carry out the attacks, several rules formatted in XML have been implemented. Within the rules it was necessary to indicate the HTTP/2 attributes to recognize, in order to filter the packets suitable for the specific attack. In addition to this, it was sometimes necessary to use embedded C functions in the XML code for processing the extracted data.

5.1 Scenario 1: Flooding Attacks

Flooding attacks, subclass of the better known DOS attacks(Denial of Service), involve the use of legitimate requests repeated thousands of times in order to saturate the server bandwidth and inhibit its correct functioning. Here, this type of attack consisted in flooding POST and GET methods, the WINDOW_UPDATE type packets or the Settings. The flooding of POST and GET methods consist in sending a large number of requests. The WINDOW_UPDATE and Settings attacks aim exploiting the streams mechanisms introduced by HTTP/2 to maintain a high number of active connections without actually using them. In addition to the aforementioned attacks, the Compression attack was implemented. This attack exploits the feature introduced by HTTP/2 to compress packets.

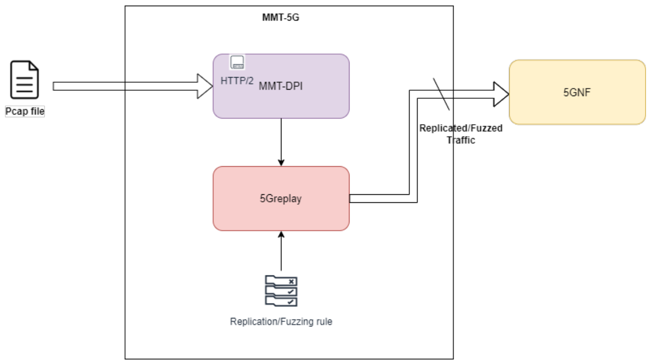
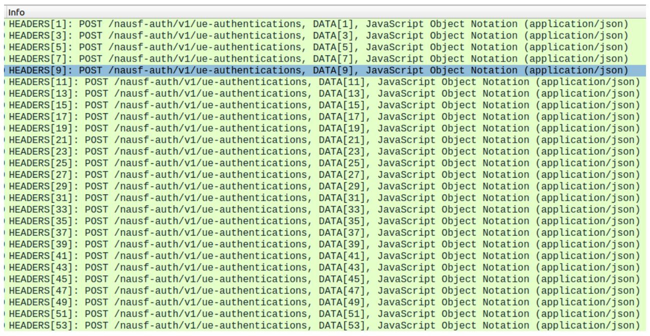


Figure 3: Flooding and fuzzing attacks in 5Greplay.

5.1.1 *Flooding of POST and GET.*

As can be seen from the image 3, a PCAP file containing the communications trace in Nokia's 5G core is provided as input to 5Greplay. The rule to replicate packets with POST and GET methods uses a C function embedded in the code in order to incrementally modify the stream ID each time. In this way, the multiplexing mechanism is exploited in order to saturate the connection with server. After that, the packet is forwarded to the server for thousands of times. The architecture of the attack is presented in Figure 3.

The result of the execution can be seen using Wireshark as shown in Figure 4. Besides this, an example of an XML rule related to DOS using POST methods is shown in Figure 5. A property identifier and a brief description of the property is given for identifying each rule. The rule is defined to filter all POST method packets using the condition `http2.header_method == 131`, where 131 is the hpack encoding for POST methods. The rule also calls the embedded function "`em_modify_then_forward`" in order to modify the stream ID of the request in the header and payload of each iteration. Through this function, users can set the number of requests to send to the server using the parameter `nb-copies` . The function also sends a WINDOW_UPDATE frame after two thousand iterations in order to enlarge the server receiving window and to avoid the rejection of the requests.



```
Info
HEADERS[1]: POST /nausf-auth/v1/ue-authentications, DATA[1], JavaScript Object Notation (application/json)
HEADERS[3]: POST /nausf-auth/v1/ue-authentications, DATA[3], JavaScript Object Notation (application/json)
HEADERS[5]: POST /nausf-auth/v1/ue-authentications, DATA[5], JavaScript Object Notation (application/json)
HEADERS[7]: POST /nausf-auth/v1/ue-authentications, DATA[7], JavaScript Object Notation (application/json)
HEADERS[9]: POST /nausf-auth/v1/ue-authentications, DATA[9], JavaScript Object Notation (application/json)
HEADERS[11]: POST /nausf-auth/v1/ue-authentications, DATA[11], JavaScript Object Notation (application/json)
HEADERS[13]: POST /nausf-auth/v1/ue-authentications, DATA[13], JavaScript Object Notation (application/json)
HEADERS[15]: POST /nausf-auth/v1/ue-authentications, DATA[15], JavaScript Object Notation (application/json)
HEADERS[17]: POST /nausf-auth/v1/ue-authentications, DATA[17], JavaScript Object Notation (application/json)
HEADERS[19]: POST /nausf-auth/v1/ue-authentications, DATA[19], JavaScript Object Notation (application/json)
HEADERS[21]: POST /nausf-auth/v1/ue-authentications, DATA[21], JavaScript Object Notation (application/json)
HEADERS[23]: POST /nausf-auth/v1/ue-authentications, DATA[23], JavaScript Object Notation (application/json)
HEADERS[25]: POST /nausf-auth/v1/ue-authentications, DATA[25], JavaScript Object Notation (application/json)
HEADERS[27]: POST /nausf-auth/v1/ue-authentications, DATA[27], JavaScript Object Notation (application/json)
HEADERS[29]: POST /nausf-auth/v1/ue-authentications, DATA[29], JavaScript Object Notation (application/json)
HEADERS[31]: POST /nausf-auth/v1/ue-authentications, DATA[31], JavaScript Object Notation (application/json)
HEADERS[33]: POST /nausf-auth/v1/ue-authentications, DATA[33], JavaScript Object Notation (application/json)
HEADERS[35]: POST /nausf-auth/v1/ue-authentications, DATA[35], JavaScript Object Notation (application/json)
HEADERS[37]: POST /nausf-auth/v1/ue-authentications, DATA[37], JavaScript Object Notation (application/json)
HEADERS[39]: POST /nausf-auth/v1/ue-authentications, DATA[39], JavaScript Object Notation (application/json)
HEADERS[41]: POST /nausf-auth/v1/ue-authentications, DATA[41], JavaScript Object Notation (application/json)
HEADERS[43]: POST /nausf-auth/v1/ue-authentications, DATA[43], JavaScript Object Notation (application/json)
HEADERS[45]: POST /nausf-auth/v1/ue-authentications, DATA[45], JavaScript Object Notation (application/json)
HEADERS[47]: POST /nausf-auth/v1/ue-authentications, DATA[47], JavaScript Object Notation (application/json)
HEADERS[49]: POST /nausf-auth/v1/ue-authentications, DATA[49], JavaScript Object Notation (application/json)
HEADERS[51]: POST /nausf-auth/v1/ue-authentications, DATA[51], JavaScript Object Notation (application/json)
HEADERS[53]: POST /nausf-auth/v1/ue-authentications, DATA[53], JavaScript Object Notation (application/json)
```

Figure 4: Packets after being replicated by 5Greplay.

5.1.2 *WINDOW_UPDATE and Setting Attacks.*

Like other protocols, HTTP/2 provides flow control to adjust the sending frequency based on the server's ability to receive messages correctly. The frame used to establish the number of bytes that the sender can send is WINDOW_UPDATE, and the maximum number established by the RFC[10] is $2^{31} - 1$. In this case, the client sends a number to indicate the maximum size of a frame that can be transmitted through the window_size_increment field. Each time that a client sends WINDOW_UPDATE, the server updates its window in order to receive more data, implying resource consumption [3, 17].

The Settings frame follows a Magic frame and is used to establish an HTTP/2 connection. In this case, the attributes of interests are SETTING_MAX_CONCURRENT_STREAM, used to set the maximum number of parallel streams that can be instantiated, and SETTINGS_INITIAL_WINDOW_SIZE, used to dimension the window of the receiver. The attack involves changing SETTING_MAX_CONCURRENT_STREAM to a high number so that the server will have to instantiate many threads, and at the same time changing SETTINGS_INITIAL_WINDOW_SIZE to a low value to slow down the responses. After the configuration phase, the attack involves sending a certain number of GET requests, interleaved by WINDOW_UPDATE frames, so that the server doesn't deny any of the requests. In this way, the bandwidth and computational resources of the server are consumed by the server.

5.1.3 *Compression Attack.*

HTTP/2 introduces a compression mechanism in order to optimize the communications and reduce the packet sizes. Although compression represents an advantage in many respects, it can be used maliciously as it consumes a lot of computing resources. In the specific case here, the attack plans to:

(1) Extract a packet from the PCAP file
(2) Decompress the packet and extract the information
(3) Add a large number of characters to the path
(4) Compress the packet again
(5) Send it to the server

```
<beginning>
<embedded_functions><![CDATA[
#include <tcpip/http2.h>
#include <stdlib.h>
static unsigned long int number_request = 0;
void on_load(){
 number_request = get_http2_env_nb_copies();
}
static void em_modif_then_forward(
 const rule_info_t *rule, int verdict, uint64_t timestamp,
 uint64_t counter, const mmt_array_t * const trace ){
 for(int i=0; i<number_request; i++){
  //stream ID must be an odd number
  set_numeric_value( PROTO_HTTP2, HTTP2_HEADER_STREAM_ID, (i
     *2+1) );
  forward_packet();
 }
}
]]></embedded_functions>
<property value="THEN" property_id="13" type_property="
     FORWARD" description="Modify stream id" if_satisfied="
     em_modif_then_forward">
  <event value="COMPUTE" event_id="1" description="Modify
     stream ID"
    boolean_expression="( http2.header_method == 131 )" />
</property>
</beginning>
```

Figure 5: 5Greplay rule for the POST flooding attack

This operation is repeated a large number of times and causes the server to waste computational resources to decompress the header, allocate memory and interpret information relating to a path that does not exist. The overhead can result in server slowdown and service degradation. The library chosen for the compression and the decompression of the headers is nghttp2, an open-source and flexible solution in C language. Nghttp2 provides a high performance implementation of HTTP/2 and broad support through accurate documentation and an active community of developers.

5.2 Scenario 2: Fuzzing Attacks

Fuzzing is a technique used to change the real data with some random values or characters, including those not allowed by the protocol specifications. It allows testing the resilience of the server. In the case here, the path characters are changed in a random way. The goal of the fuzzing is to test the resilience of the server to invalid data. In the context of 5Greplay, two fuzzing attacks have been developed.

In the first attack, the headers of the POST packets are decompressed. For this, the information relating to the path is extracted and the data included in path header field is changed with random characters. After modification, the package is compressed and sent to the server. This kind of attack is more difficult to recognize since a decompression must be performed before checking for illegal characters.

The second attack involves modifying the content of the payload. In particular, information about the JSON files is changed within the POST packets, and legitimate data is modified using random characters, including some that are not allowed by the JSON specification. The set of characters involved in this attacks includes the letters of the alphabet, numbers, symbols like '%',' \' and so on. These latest attacks join other fuzzing attacks developed for 5Greplay and described in [12], and target 5G core functions(e.g., AMF, SMF). In order to access to payload data the plugin in MMT-DPI accesses to the header length and adds this value to the HTTP/2 protocol offset and returns a pointer.5Greplay uses the pointer and the payload length previously extracted to modify the data as described before.

In addition to the offensive security tools, Montimage provides a very effective tool for attack detection called MMT-Security. The tool takes rules written in XML language as input to raise alerts. In this regard, three rules have been developed: one for recognizing DOS attacks, one for recognizing Compression attacks and the last one for recognizing Fuzzing attacks. The rule for DOS recogni-

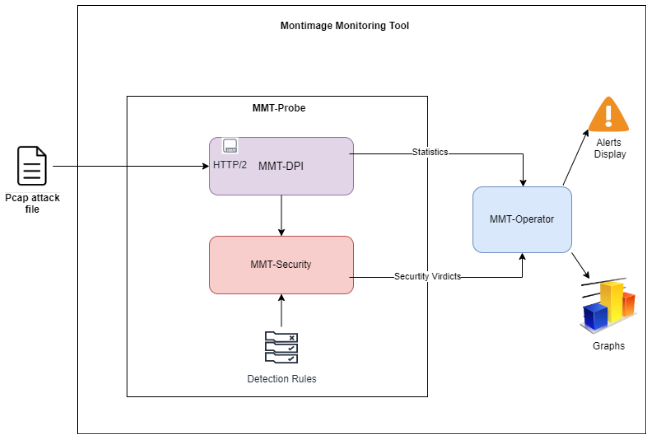
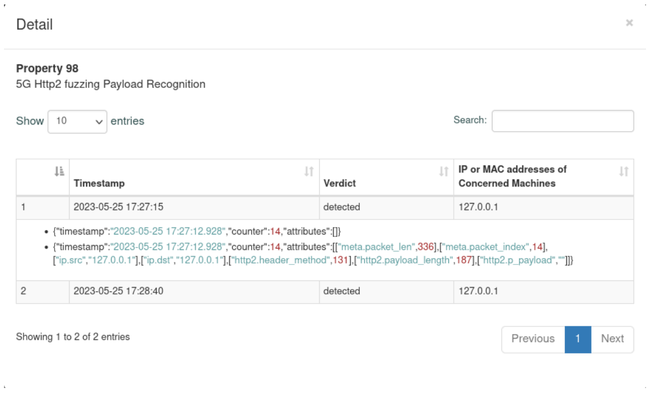


Figure 6: Montimage Monitoring tool Architecture

tion counts how many POST/GET requests or WINDOW_UPDATE packets are sent in a given time range, e.g., between 0 and 100 ms. An alert is raised if the packets counted exceed the user-defined threshold. This threshold must be suitably sized since an exceedingly large number may not raise alerts when an attack occurs, while a too small number may raise alerts even during correct conditions. The rule for Compression attacks checks the dimension of the header. If the header size exceeds a threshold, an alert is raised. Finally, the rule for detecting fuzzing checks if there are any unusual characters inside the payload.

The MMT-Probe processes the network traffic using the MMT-Security library and rules, and sends extracted data and alerts to an application (MMT-Operator) that processes them in order to create statistics and notifications. MMT-Probe output is a sequence of files in JSON or CVS format containing information about events that have been observed during a user-configurable period of time. MMT-Operator is a web application with a user-friendly GUI that displays the statistics using graphs and alerts in different dashboards, and allows managing the MMT-Probes. Figure 6 shows the architecture of Montimage Monitoring Tool. Figure 7 shows an alert related to a Fuzzing attack.



Property 98
5G Http2 fuzzing Payload Recognition

	Timestamp	Verdict	IP or MAC addresses of Concerned Machines
1	2023-05-25 17:27:15	detected	127.0.0.1
2	2023-05-25 17:28:40	detected	127.0.0.1

```
• {"timestamp":"2023-05-25 17:27:12.928","counter":14,"attributes":[]}
• {"timestamp":"2023-05-25 17:27:12.928","counter":14,"attributes":[["meta.packet_len",336],["meta.packet_index",14],
  ["ip.src","127.0.0.1"],["ip.dst","127.0.0.1"],["http2.header_method",131],["http2.payload_length",187],["http2.p_payload",""]]}
```

Showing 1 to 2 of 2 entries

Figure 7: Fuzzing attack detection in MMT-Operator

5.3 Results

The table 1 resumes the tests using 5Greplay, where the second column specifies the number of messages sent to the SUT (System Under Test) and the third column indicates the number of alerts raised. When traffic is replayed, HTTP/2 tends to aggregate multiple requests in the same packet, exploiting the mechanism of TCP segmentation. For this reason, sometimes the packet sizes increase, and MMT-Security rule raises the following warning message: "WARN_SEC: Report 1262 for 700.4098 is too big (req. 65477, avail. 33206 bytes), must increase "input.max_message_size". Despite this, alerts are correctly generated. The threshold for DOS detection is set to 50 packets in the time range between 0 and 100 ms.

Table 1: Result of the detection.

Attacks	Messages	Alerts
Setting attack	1272	62
Flooding of GET	173	19
Flooding of POST	168	21
Compression attack	100	60
Flooding of WINDOW_UPDATE	28852	28440
Path Fuzzing	14175	2519
Payload Fuzzing	113	62

6 CONCLUSION

This paper analyzes the security issues regarding HTTP/2 adoption for 5G networks. The introduction of HTTP/2 increases the 5G performance and flexibility, but also introduces new security issues. In this paper we propose security testing methodologies that have been implemented in an open-source tool, 5Greplay. This tool allows testing 5G network components using fuzzing and attack injection techniques. To illustrate this, this paper analyzes the different attacks introduced by HTTP/2, implements them, and shows how they can be detected by MMT. It is used in the SANCUS framework offering network operators with an integrated solution for the detection, prevention, protection and response against potential attacks, the safeguard of privacy, and network service continuity.

ACKNOWLEDGMENTS

This work was made possible with funding from the European Union's Horizon 2020 research and innovation program, under grant agreement No. 952672 (SANCUS). The opinions expressed and arguments employed herein do not necessarily reflect the official views of the funding body.

REFERENCES

[1] M. Liyanage J. Okwuibe M. Ylianttila . I. Ahmad, T. Kumar and A. Gurtov. 2018. Overview of 5g security challenges and solutions.
[2] M. Liyanage J. Okwuibe M. Ylianttila . I. Ahmad, T. Kumar and A. Gurtov. Feb 2022. Enisa threat landscape for 5G networks.
[3] Baig Z. Lam C.P. Hingston Adi, E. 2015. Low-rate denial-of-service attacks against http/2 services. *Security and Privacy in Communication Networks* (2015), 1–5.
[4] Jin Cao, Maode Ma, Hui Li, Ruhui Ma, Yunqing Sun, Pu Yu, and Lihui Xiong. 2020. A Survey on Security Aspects for 3GPP 5G Networks. *IEEE Commun. Surv. Tutorials* 22, 1 (2020), 170–195. https://doi.org/10.1109/COMST.2019.2951818
[5] Efstratios Chatzoglou, Vasileios Kouliaridis, Georgios Kambourakis, Georgios Karopoulos, and Stefanos Gritzalis. 2023. A hands-on gaze on HTTP/3 security through the lens of HTTP/2 and a public dataset. *Comput. Secur.* 125 (2023), 103051. https://doi.org/10.1016/j.cose.2022.103051
[6] Ning Zhang Fatima Salahdine, Tao Han. 2023. Security in 5G and beyond recent advances and future challenges. *Security Privacy.* (2023). https://doi.org/10.1002/spy2.271
[7] Xinxin Hu, Caixia Liu, Shuxin Liu, Wei You, and Yu. Zhao. 2018. Signalling security analysis: Is HTTP/2 secure in 5G core network? *Security and Privacy in Communication Networks* (2018).
[8] Z. Mukhtar2 B. Yahaya2 Y. Ibrahim2 M. O. Momoh K. O. Shobowale1, *. 2023. Latest Advances on Security Architecture f or 5 G Technology and Services. *INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING COMPUTER SYSTEMS (IJSECS)* 9, ISSUE 1 (2023), 27 – 38. https://doi.org/10.15282/ijsecs.9.1.2023.3.0107
[9] Rabia Khan; Pardeep Kumar; Dushantha Nalin K. Jayakody; Madhusanka Liyanage. 2020. A Survey on Security and Privacy of 5G Technologies: Potential Solutions, Recent Advancements, and Future Directions. *IEEE Communications Surveys Tutorials* 22 (2020).
[10] Martin Thomson Mike Belshe, Roberto Peon. 2015. RFC Hypertext Transfer Protocol Version 2 (HTTP/2). https://datatracker.ietf.org/doc/html/rfc7540
[11] Montimage. 2023. 5GReplay Documentation. http://5greplay.org/
[12] Zujany Salazar, Huu Nghia Nguyen, Wissam Mallouli, Ana R. Cavalli, and Edgardo Montes de Oca. 2021. 5Greplay: A 5G Network Traffic Fuzzer - Application to Attack Injection. In *Proceedings of the 16th International Conference on Availability, Reliability and Security (ARES 21).* Article 106, 8 pages. https://doi.org/10.1145/3465481.3470079
[13] Zujany Salazar, Fatiha Zaïdi, Huu Nghia Nguyen, Wissam Mallouli, Ana Rosa Cavalli, and Edgardo Montes de Oca. 2023. A Network Traffic Mutation Based Ontology, and Its Application to 5G Networks. *IEEE Access* 11 (2023), 43925–43944. https://doi.org/10.1109/ACCESS.2023.3268759
[14] T. 2020. 3rd Generation Partnership Project (3GPP) 3gpp ts 33.117. *catalogue of general security assurance requirements* (2020).
[15] Silvio Valenti, Dario Rossi, Alberto Dainotti, Antonio Pescapè, Alessandro Finamore, and Marco Mellia. 2013. *Reviewing Traffic Classification.* Springer Berlin Heidelberg, Berlin, Heidelberg, 123–147. https://doi.org/10.1007/978-3-642-36784-7_6
[16] Nathalie *Wehbe, Hyame Assem Alameddine, Makan Pourzandi, Elias Bou-Harb, and Chadi. Assi. [n. d.]. Assessment of HTTP/2 Usage in 5G Service Based Architecture. *IEEE Communications Magazine* ([n. d.]).
[17] Shouling Ji Meng Han Xiang Ling1, Chunming Wu1. 2023. H2DoS: An Application-Layer DoS Attack Towards HTTP/2 Protocol. *Security and Privacy in Communication Networks* (2023), 550–570. https://link.springer.com/chapter/10.1007/978-3-319-78813-5_28