Poisoned GOOSE: Exploiting the GOOSE Protocol

Nishchal Kush[1] **Ejaz Ahmed**[2] **Mark Branagan**[3] **Ernest Foo**[4]

Information Security Discipline,
Queensland University of Technology,
Email: {`n.kush`[1], `m.branagan`[3], `e.foo`[4]}`@qut.edu.au`
Email: `ejaz9645@gmail.com`[2]

Abstract

This paper presents a vulnerability within the generic object oriented substation event (GOOSE) communication protocol. It describes an exploit of the vulnerability and proposes a number of attack variants. The attacks sends GOOSE frames containing *higher* status numbers to the receiving intelligent electronic device (IED). This prevents legitimate GOOSE frames from being processed and effectively causes a hijacking of the communication channel, which can be used to implement a denial–of–service (DoS) or manipulate the subscriber (unless a status number roll-over occurs). The authors refer to this attack as a *poisoning* of the subscriber. A number of GOOSE poisoning attacks are evaluated experimentally on a test bed and demonstrated to be successful.

Keywords: substations, GOOSE protocol, critical infrastructure security

1 Introduction

Generic object oriented substation event (GOOSE) is a part of the International Electrotechnical Commission (IEC) 61850 (International Electrotechnical Commission 2005) suite of standards and specifies the communication of electrical substation events. IEC 61850 is integral to enabling interoperable substation automation systems (SASs). Interoperable SASs are a key component of the smart grid. GOOSE is multicast on the data–link layer and commonly deployed over fibre-optic or shielded twisted pair cable to relay event information.

In this work, a vulnerability in the GOOSE protocol is identified. The handling of status numbers in GOOSE frames provides the opportunity to implement attacks. These attacks can be used to hijack the communication with the subscriber to prevent legitimate GOOSE messages from being processed and to spoof additional attack traffic to manipulate the subscriber.

Depending on the functionality of the specific compromised subscriber, the impact can be significant. For example, if the compromised subscriber was for the operation of electrical protection. Its operations could be adversely affected, resulting in safety issues, such as, damage to the electrical network or injury to humans.

The contributions of this paper are the analysis of the GOOSE protocol to identify vulnerabilities in the processing algorithm and the implemention of practical attacks to exploit this vulnerability. The attacks are presented using a test bed comprising a virtual publisher, subscriber and attacker, developed in Java. The attacks were successful in all cases.

The remainder of the paper is structured as follows; this section presents a brief background and discussion of the GOOSE protocol. Section 2 summarises the proposed attacks. Section 3 presents the methodology employed in this research and describes the experiments performed. Section 4 presents the results of the experiments. The results are evaluated and discussed in Section 5. Finally Section 6 concludes the paper.

Background

Since GOOSE is multicast using the data–link layer, there is no logical address and flow control functionality. Thus, there is no support for message authentication. Part 6 of the IEC 62351 (International Electrotechnical Commission 2007*b*) standard defines message security mechanisms for GOOSE. However, due to the strict time performance requirements of GOOSE messages "security measures which affect transmission rates are not acceptable." (International Electrotechnical Commission 2007*a*, p.30)

GOOSE messages allow a sending intelligent electronic device (IED), a *publisher*, to multicast user–configurable state data to receiving IEDs, known as *subscribers*. GOOSE is an unacknowledged connectionless communication protocol, in that, the subscribers do not send an acknowledgement to the publishers. Substation *events* cause IEDs to transmit a GOOSE message. GOOSE messages contain a *status number* and *sequence number* amongst other data. These messages are retransmitted with increasing delay and sequence number until the next event, which requires the status number to be incremented. Status numbers are intended to provide replay protection (International Electrotechnical Commission 2007*b*). Both the status number (*stNum*) and sequence number (*sqNum*), are represented as 32 bit unsigned integers, thus having a possible value range from 0 to 2^{32}.

Once a GOOSE frame is received, Algorithm 1, as derived from the IEC 62351 standard (International Electrotechnical Commission 2007*b*), is employed. A GOOSE frame with a lower status number than that of the previously received message is not processed unless there has been a status number roll–over or time–out.

The following section describes the proposed attacks which exploit a vulnerability in the GOOSE

Algorithm 1: GOOSE processing algorithm

```
1  previous stNum ← get the previously processed stNum;
2  message StNum ← get stNum from message;
3  if  message StNum $\neq$ previous StNum then
4      if  message StNum < previous StNum AND no stNum roll-over AND no TTL time-out then
5          discard message;
6      end
7      else if  stNum roll-over OR no TTL time-out then
8          re-establish stNum;
9      end
10     age ← current time-stamp - time-stamp on message;
11     if | age | > 2 minimum skew then
12         discard message;
13     end
14     else
15         process message;
16     end
17 end
18 else
19     discard message;
20 end
```

protocol.

2 Proposed Attacks

The hypothesis of this paper is that a malicious GOOSE message could hijack the communication between a subscriber and publisher, and could be used to prevent the subscriber from processing subsequent legitimate GOOSE messages or to influence the subscriber by forcing it to process fabricated GOOSE messages. This will be the case where legitimate GOOSE messages have status numbers equal to or less than the status number in the fabricated message. This attack is referred to, by the authors, as GOOSE *poisoning*. Three variants of this attack have been summarised below;

- High Status Number Attack – The first variant multicasts a single spoofed GOOSE frame with a very high status number to a subscriber after inspecting GOOSE frames. It is expected that once the spoofed GOOSE frame is processed, any legitimate GOOSE frames, with status number equal to or less than this will not be processed by the subscriber.

- High Rate Flooding Attack – The second variant, requires the attacker to multicast a range of spoofed GOOSE messages, with increasing status numbers, after inspecting an initial GOOSE frame. The high rate flooding of spoofed packets is expected to eventually employ a status number that is higher than the expected status number on the subscriber. This variant is best summarised as a status number *flooding* attack.

- Semantic Attack – The third and final variant, is a Semantic Attack. This attack is executed in two phases. The first phase requires the attacker to observe the network traffic and inspect GOOSE messages to determine the status numbers in use and infer the rate of status change. The second phase requires the attacker to extrapolate an attack rate, which is higher than the observed rate. Spoofed GOOSE messages are then multicast at the attack rate with status numbers incrementing by one. The attack traffic is expected to *race* against legitimate traffic and prevent bonafide GOOSE messages from being processed by the subscriber.

The three variants aim to exploit the lack of authentication and confidentiality at the data–link layer. The lack of authentication enables the spoofed GOOSE frames to be processed. The lack of confidentiality enables the content of legitimate GOOSE frames to be read. The attacks are expected to effectively hijack the communication, and thus the processing, of GOOSE messages on the subscriber. In all three variants the subscriber is forced to process a higher status number than the status number employed by the legitimate publisher. As a result the subscriber does not service legitimate GOOSE frames and can subsequently be controlled by the attacker.

Attacker Model

Given the nature of the GOOSE protocol, the adversary is assumed to have access to the data–link layer of the substation local area network (LAN). The attacker model used for the proposed attacks does not require the adversary to have direct access to the publisher nor subscriber. Instead the adversary must be capable of analysing and spoofing GOOSE frames. The attacker must be able to spoof media access control (MAC) addresses in order to inject GOOSE frames that appear to originate from a legitimate publisher.

The experimental methodology, including the software simulation and test bed setup to evaluate the proposed attacks are described in the following section.

3 Methodology

To validate the hypothesis and execute the described attacks a simple experimental test bed was established. The test bed provided a controlled and isolated environment for experimentation. The test bed was designed to simulate a specific scenario, and emulate the expected network traffic. The simulation scenario, its implementation, the test bed setup, and the experiments conducted are discussed below.

3.1 Scenario

Transmission substations usually incorporate transfer tripping and bus bar protection, along with circuit breakers and switches. The simulation scenario employed for the experiments focused on a single circuit breaker. Starting at $t = 0\ s$ the publisher multicasts GOOSE messages. The initial status number used was 5891. The multicast message includes a *Boolean* indicating a circuit breaker state, and a *float* representing a simulated voltage reading on the bus bar. At $t = 25\ s$, a status change event is simulated to cause a *trip* GOOSE message to be multicast, i.e. Boolean is set to *true*. The trip event is simulated for 4 s, and at $t = 29\ s$, the simulated publisher is reverted to its original state, i.e. Boolean is set to *false*, until the simulation terminates at $t = 60\ s$. During the simulation scenario, the subscriber logs the state of the circuit breaker to indicate if it's open or closed, based on the Boolean value in the GOOSE frame.

The subscriber log file entry included a time stamp represented as *epoch* and a *string* message indicating the circuit breaker state, the processed frame status number, the sequence number, the frame received epoch, the frame processed epoch and the difference between the received and processed epoch times. An excerpt from the log is presented in Figure 1.

3.2 Simulation

A virtual publisher and subscriber were developed in Java using the *jnetpcap* software library. The library is a wrapper for the *libpcap* library to implement low–level network data injection. The publisher was designed to multicast GOOSE frames as described in Section 3.1. A simulation engine was also developed to control the execution of the publisher. The simulation engine was used as a harness to start and stop the publisher. The publisher multicasts a GOOSE frame every 50 ms, and increments the status number every 100 ms. The simulation models a stable environment, i.e. the frequency and duration of events are regular. In an unstable environment these events are irregular depending on environmental conditions. The subscriber was configured to subscribe to the multicast GOOSE frames from the publisher. The subscriber implemented the processing algorithm, Algorithm 1, and was programmed to log the received GOOSE messages if they were processed. A single attacker was used to implement the three variants of the attack. The attacker accepted user input to determine the attack type.

3.3 Test Bed Setup

The test bed consists of a publisher, a subscriber, and an attacker connected via a network switch. An overview of the test bed is presented in Figure 2.

Both, the publisher and subscriber were hosted on Dell Optiplex 990 desktop machines, with Intel Core i7-2600 processors running at 3.4Ghz and 8GB of RAM. The attacking host was a Dell Optiplex 960 desktop machine with Intel Core2 Duo E8400 processor running at 3.0Ghz and 4GB of RAM. All network hosts were running the 64–bit version of CentOS 6.3 operating system (OS), and used the same Intel PRO/1000 network interface card (NIC).

These machines were connected via Fast Ethernet to a Cisco 2950T managed network switch with enhanced software image (IOS C2950-I6K2L2Q4-M). The network switch is intended to emulate the substation bay switch within a substation. The switch was configured with a monitor port to enable the capture of ingress and egress traffic of the ports connecting the publisher and subscriber. A separate Broadcom Tigon3 NIC was used on the subscriber to connect to the monitor port and capture traffic using the `tcpdump` utility. The subscriber is also set-up as a network time protocol (NTP) time server to provide time synchronisation on the network.

The following section describes the experiments using the implemented attacks on the test bed.

```
1360651521968 INFO: XCBR closed 5891 0
1360651521968 1360651521968 0
1360651522092 INFO: XCBR closed 5892 0
1360651522092 1360651522092 0
1360651522218 INFO: XCBR closed 5893 0
1360651522218 1360651522218 0
```

Figure 1: Excerpt from virtual subscriber log file

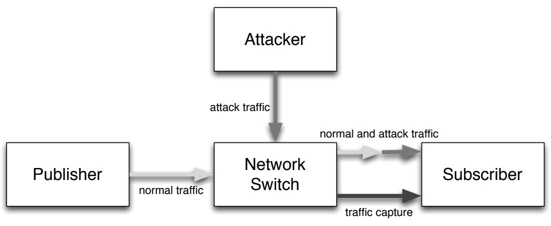

Figure 2: Test bed setup

3.4 Experiments

The proposed attacks as described in Section 2 were executed in the isolated environment provided by the test bed. Each experiment was executed independently, i.e. the virtual publisher and subscriber are restarted following each experiment. Before each experiment the network capture packet capture (PCAP) and log files were cleared. Following each experiment, the files are extracted and saved for analysis. The experiments are summarised below;

- No Attack – an experimental control to observe normal simulated behaviour. In this experiment no attacks were implemented. The simulation scenario is executed unhindered. In the control, the circuit breaker frames were successfully received, processed, and logged. The results of this control were used to determine if attacks were successful.

- High Status Number Attack – A single fabricated GOOSE message with status number close to 2^{32} was transmitted. The maximum value was not used as it would cause a roll–over, instead the value $2^{32} - 1$ was used. Successful poisoning of the subscriber using any other 32 bit value would have a probability of $\frac{1}{2^{32}}$, while the value used in the experiment has a probability of success of $\frac{2^{32}-2}{2^{32}}$. It was expected that once the fabricated GOOSE frame was processed by the subscriber, the subscriber stopped processing legitimate GOOSE messages with lower status numbers.

- High Rate Flooding Attack – In the High Rate Flooding Attack, GOOSE frames are fabricated with increasing status numbers. In this experiment status number of 5800 was used. This value was selected due to the limited duration of the experiment. GOOSE frames should be multicast as rapidly as possible by the attacker. In the case of the experimental test bed this rate was 1000 frames per second. It was expected that once the attack traffic started being processed by the subscriber, legitimate traffic with lower status numbers would be discarded.

- Semantic Attack – The third and final experiment is described as a Semantic Attack. During the first phase of the Semantic Attack, the attacker observed three legitimate GOOSE frames and recorded the highest status number used. The average inter-frame delay was calculated based on the observed traffic. In the second phase of the attack, a lower attack delay was used to calculate an attack rate. The attacker then spoofed and multicast attack traffic with increasing status numbers at the calculated attack rate. As with the High Rate Flooding Attack, it was expected that once the fabricated frames were processed, legitimate traffic with lower status numbers would be discarded.

The network traffic and subscriber log files were recorded and analysed for each experiment. The following section presents the experimental results.

4 Results

The results for each experiment comprised the network capture PCAP file and the virtual subscriber log file. The log file was processed to indicate the circuit breaker state at various timestamps using a custom shell script. The captured network traffic PCAP file was processed using another custom utility (`goosestat`) written in Java to extract specific features, such as, the status numbers, sequence numbers, payload data, and frame time stamp. An excerpt from the processed file is presented in Figure 3;

To better understand the results, the concept of *convergence* was introduced. Convergence in the context of GOOSE poisoning attack is when the attacker and the legitimate publisher are transmitting GOOSE frames with the same status number. Once the attacker exceeds this status number GOOSE poisoning can occur. Convergence only occurs if the attack GOOSE frame rate (λ_A) is higher than the GOOSE status number change frame rate (λ_N) of the legitimate publisher.

A second concept required for understanding experimental results is *sending advantage*. Sending advantage (α) is defined as the difference in the status numbers employed by the attacker ($stNum_A$) and the legitimate publisher ($stNum_N$) at the point the attack begins. Let α be some unknown sending advantage that a legitimate publisher has over the attacker, then time till convergence can be calculated using (1) once the attack starts.

$$t = \frac{\alpha}{\lambda_A - \lambda_N} \tag{1}$$

where:

- t: convergence time in seconds
- α: difference between legitimate and attack status numbers (sending advantage)
- λ_A: attack rate in frames per second
- λ_N: estimated average normal traffic status number change rate in frames per second

```
#
# goosestat version 0.2
# goosestat -r no-attack.pcap -p
#
# frame stNum sqNum payload data timestamp
rtimestamp
#
1 5891 0 0 1360651521966 19388
2 5891 1 0 1360651522028 19450
3 5892 0 0 1360651522091 19513
4 5892 1 0 1360651522154 19576
5 5893 0 0 1360651522217 19639
6 5893 1 0 1360651522279 19701
```

Figure 3: Except from output of goosestat on PCAP file

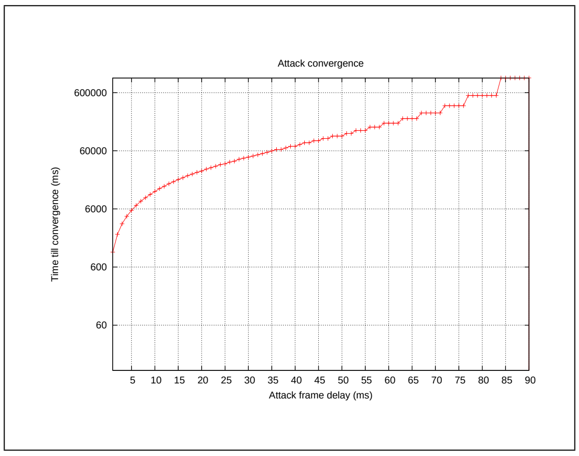

Figure 4: Convergence time for simulated scenario

The sending advantage (α) that a legitimate publisher has over an attacker is variable. Since $stNum_N = stNum_A + \alpha$, in the best case, from the attackers perspective, the legitimate status number is equal to the attackers status number, i.e. $\alpha = 0$. In the attackers worst case, the legitimate status number, is close to half the maximum, i.e. $\alpha \approx \frac{2^{32}}{2} \approx 2^{31}$, instead of $\alpha \approx 2^{32}$ because of status number roll–over. If the legitimate sender is at the maximum status number, then subsequent frames would cause a roll–over of the status number, thereby rendering the next attack frame transmission as *poisonous*. Hence the sending advantage would become negative.

To estimate the convergence time for the simulated scenario, an average case sending advantage of $\alpha = \frac{2^{31}}{2}$ was used. Figure 4 presents time till convergence based on the simulation scenario for the average case sending advantage where a virtual publisher simulates a 100 ms inter-frame delay. The figure illustrates varying *attack frame delays* employed by the attacker and the expected *time till convergence*. Considering the average case, an attacker could converge upon the status number being employed by the legitimate publisher, and subsequently start poisoning the subscriber within 6000 ms if using an attack rate greater than 200 frames per second.

The results obtained from the experiments are discussed in the following section.

5 Discussion

The success of the attacks were ascertained by comparing the attack results with the control results. The attack was considered successful, when the subscriber failed to correctly log the circuit breaker state. In Figures 5,6,7 and 8, the change in the status number with respect to the simulation time, is plotted on the y–axis from the PCAP file content. The simulation time, calculated relative to the first frame, is plotted on the x-axis. On the y2-axis, the state of the circuit breaker is plotted from the subscriber log file data. Again, the simulation time was calculated relative to the first log entry. A state of 1 meant the circuit breaker was closed, and 0 indicated it was open.

The scenario employed for the simulation has been simplified for the purposes of brevity and rapid development. As previously described, a stable system was used in the simulation. Further the convergence calculations employed the average status number change rates. These simplifications may provide a skewed

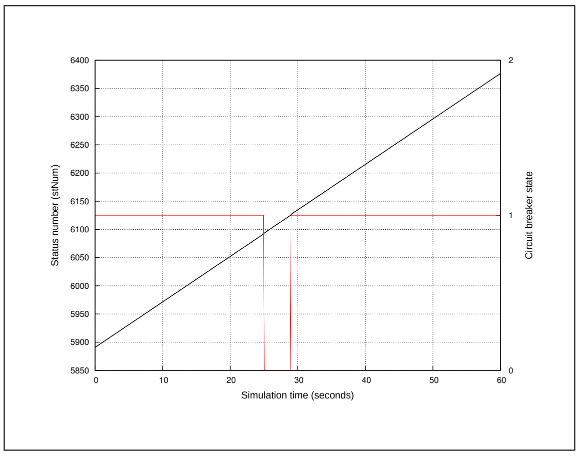

Figure 5: Simulation run without attacks

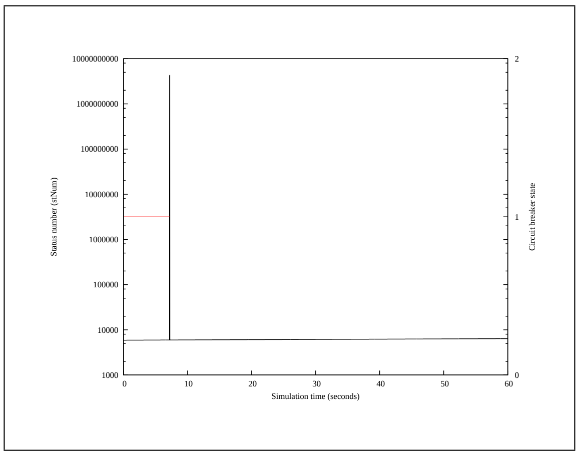

Figure 6: High Status Number Attack

perception of real GOOSE traffic in an unstable system. The results obtained are summarised below;

- No Attack – As illustrated in Figure 5, the control illustrates the simulation scenario precisely, i.e. the circuit breaker state is changed to open at *t = 24 s* and remains in this state until *t = 29 s*, and reverts to a closed state for the remainder of the simulation.

- High Status Number Attack – In the High Status Number Attack, once the attack frame with the large status number was processed by the subscriber, all other legitimate GOOSE frames were discarded. This is illustrated in Figure 6 as the lack of circuit breaker status logging by the subscriber. It should be noted that logarithmic scaling is employed on the y–axis of the figure. For the purpose of the experiment the attack is considered a success since the subscriber was unable to log the circuit breaker state, from approximately *t = 7 s*.

- High Rate Flooding Attack – successful convergence during this attack depends on the sending advantage that the legitimate publisher has over the attacker. In this attack, a successful convergence is observed. The point of convergence is illustrated in Figure 7 where the two lines intersect, at approximately *t = 8 s*. The attack illustrated uses an initial attack value of 5,800, i.e.

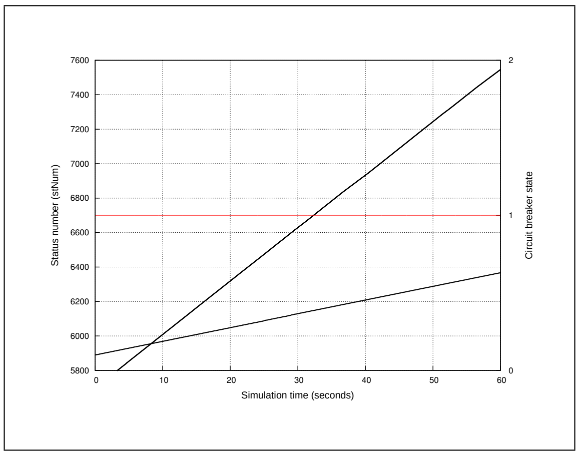

Figure 7: High Rate Flooding Attack

the advantage $\alpha = 91$. The line with the higher gradient represents the attack traffic. There was no way to distinguish the attack traffic from the legitimate GOOSE traffic save for the rate, i.e. the attack traffic rate is higher than the normal traffic rate. This attack is considered a success, since the subscriber was unable to detect the trip message from the legitimate publisher.

- Semantic Attack – In Figure 8 the point of convergence is less obvious than the flooding attack as in Figure 7. In the Semantic Attack, convergence occurred almost immediately upon start of the attack, at approximately *t = 4 s*. As with the flooding attack, there was no means to distinguish the attack traffic from normal traffic based on features of the traffic, apart from the rate at which the traffic is generated. The primary advantage of the Semantic Attack over a High Rate Flooding Attack, was that the legitimate IED is extended a lower sending advantage. Since the attacker is able to observe the network traffic and thus determine the legitimate status number in use, the sending advantage is eliminated. Another benefit of the Semantic Attack over a High Rate Flooding Attack was that a lower attack traffic rate can be used, i.e. an attack traffic rate that is a close approximation of the average traffic rate. This attack is considered a success since the subscriber does not the log the trip message from the legitimate publisher.

In all three attacks the circuit breaker trip was not logged. The poisoning is successful as long as the legitimate publisher's status number is less than the attack status number.

In the case of the High Status Number Attack (see Figure 6) the subscriber failed to log the circuit breaker state. It is expected that this attack would be successful in both stable and unstable systems. This attack does not require convergence and only a single frame is required for the poisoning. The subscriber remains poisoned until the legitimate publisher reaches a status number higher than the one used in the poisoning.

In the High Rate Flooding and Semantic Attacks (see Figures 7 and 8 respectively) the subscriber logged an incorrect state based on the spoofed traffic. Since the High Rate Flooding and the Semantic Attacks rely on convergence, they are suited to stable environments. The subscriber remains poisoned

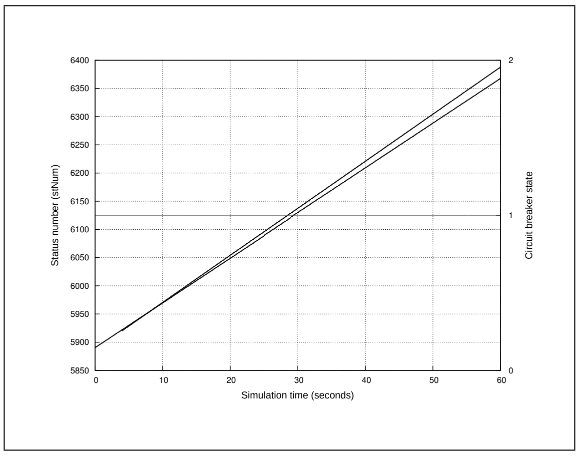

Figure 8: Semantic Attack

so long as the attacker continues the attack after convergence. The advantage of the Semantic Attack over the High Rate Flooding Attack is that convergence occurs almost immediately.

Related Work

Hoyos et al. (Hoyos et al. 2012), demonstrated an attack exploiting the GOOSE protocol. The attack first performed a capture of the network traffic for GOOSE messages, and subsequently altered the status number and *flipped* any Boolean data i.e. if the Boolean data was set to true, the attack resets it to false and vice versa. The GOOSE frame was then re-transmitted with increasing status numbers. The tampered data was intended to have an effect on the subscribing IEDs.

Unlike Hoyos et al., the attacks presented in this paper do not attempt to manipulate the content of messages to influence the subscribers actions. Instead, the attacks proposed in this paper, hijack the communication channel by exploiting a flaw in the algorithm used for GOOSE processing. Once the attacks are successful, legitimate traffic is no longer processed by the subscriber. The only messages that can be processed are those injected by the attacker. Following the hijacking, continuing to inject attack traffic effectively performs a denial–of–service (DoS) attack by denying the legitimate sender traffic access to the subscriber. The attacker could also manipulate the actions of the subscriber by, for example, replaying old payloads or altering the payload of existing traffic.

There is an existing set of attacks against the transmission control protocol (TCP) using prediction of TCP sequence numbers (Morris 1985), for a given TCP session. GOOSE status numbers share the same linear properties as TCP sequence numbers. Therefore, the GOOSE status number makes the GOOSE protocol vulnerable to similar *off-path* attacks.

The use of randomisation in the TCP sequence number selection (Gont & Bellovin 2012) made TCP sessions more resistant to sequence number estimation. However, the randomisation approach is still susceptible to man–in–the–middle (MITM) attacks. The use of randomisation of sequence numbers in TCP relies on the connection oriented nature of TCP. As GOOSE is a connectionless protocol, for randomisation to be employed would require out–of–band synchronisation of status numbers. Such out–of–band synchronisation may be difficult to achieve in the constrained substation environment.

6 Conclusion

The work presented in this paper exploits a vulnerability in the GOOSE protocol and presents three variants of an attack. The vulnerability arises as a result of predictable status number used in GOOSE messages and the processing of these messages by the subscriber. The paper demonstrated successful poisoning of the subscriber which prevented it from processing legitimate GOOSE messages.

There are a number of areas of future work arising from this paper. The semantic attack can be further developed to be adaptive, the attack rate could change based on the observed rate of legitimate traffic. In concert with developing the attack potential detection techniques also need investigation. Presently the authors are in the initial phases of implementing the attacks using actual IEDs rather than simulations. Preliminary results indicate that vendors may have an inconsistent interpretation of the IEC 61850 and IEC 61235 standards. The impact that this may have on the success or otherwise of the attacks described in this paper are not yet fully determined. Finally, the ultimate aim of this work is to develop mitigation strategies against these attacks. While mitigation strategies for the attacks have not been identified in this paper the authors are presently investigating potential mitigation strategies.

References

Gont, F. & Bellovin, S. M. (2012), ‘Defending against Sequence Number Attacks’, RFC 6528 (Proposed Standard).

Hoyos, J., Dehus, M. & Brown, T. X. (2012), Exploiting the GOOSE Protocol: A Practical Attack on Cyber-infrastructure, *in* ‘Proceedings of the IEEE Workshop on Smart Grid Communications: Design for Performance’, pp. 1508–1513.

International Electrotechnical Commission (2005), ‘IEC 61850:2005 – Communication networks and systems in substations’.

International Electrotechnical Commission (2007*a*), ‘IEC 62351-1 : Power Systems Management and Associated Information Exchange – Data Communications Security: Part 1 – Communication Network and System Security – Introduction to Security Issues’.

International Electrotechnical Commission (2007*b*), ‘IEC 62351-6 : Power Systems Management and Associated Information Exchange – Data Communications Security: Part 6 – Security for IEC 61850’.

Morris, R. T. (1985), ‘A Weakness in the 4.2 BSD Unix TCP/IP Software’, *Computer Science Technical Report No 117* .