Demand Response Capabilities Position Paper

UL Identity Management & Security
Dr. Johannes Bauer <johannes.bauer@ul.com>

September 6th 2019

Position Statement

Our power grid is rapidly changing. Gone are the days in which both production and consumption of power has been relatively easy to predict. With renewable energy as a producer and ubiquitous high-power equipment in many households, we now face a landscape in which the grid frequency stability is more difficult to maintain. While previously all this high-power equipment was in the control of network operators and energy suppliers, it now has decentralized and moved into the control of consumers. Without Demand Response Capabilities, these devices are out of reach and grid operators can only passively react to whatever is happening downstream. Therefore, it is our position that Demand Response Capabilities is a welcome and inevitable evolution of our power grids – while at the same time we see greater risks than ever before when part of critical infrastructure is now in physical control of consumers, with unlimited time to attempt to tamper with them. We support Demand Response Capabilities, but argue that if they are implemented, it is imperative they be implemented securely.

With security set as a design goal, a different conundrum arises at the same time: what constitutes an appropriate security for the level of risk involved? This is not just a technical question, but also a question that goes into lifecycles of products and their maintenance. New vulnerabilities are discovered in an alarming rate for even the most sophisticated systems. It would be naïve to assume that new systems can be built which do not have any of these issues. Therefore, the question of proper handling and mitigation is one that needs to be answered to allow for secure ongoing operation of a smart grid.

Considerations

We see several considerations that need to be addressed when discussing Demand Response Capabilities: What is the actual threat and risk, which type of security – technically and process-oriented – is appropriate and how can compliance to good security be ensured and proven by testing. In the following, we discuss them.

Threat Model

Breaking the security of a single consumer-device is likely only a minor annoyance for the affected consumer. Imagine we, the attackers, have full control over one particular HVAC system. We can remotely turn on heating or cooling at will, causing the resident of their home to lose some sleep as a worst-case scenario. As long as the affected devices have built-in safety protection and the attacker simply has no way to turn heating up to a point where the system would catch fire – something that is a standard safety requirement these days – and as long as the residents are at good health, there is no way to inflict bodily harm from the outside. Note, however, that for small children and elderly people this might be a different scenario entirely, which much higher stakes, but for the sake of brevity we will omit this discussion.

The issue with decentralized control systems is therefore not primarily attacks on individual devices, it is rather attacks that *scale well*, i.e., attacks on a great number of devices at once. Attackers in this scenario do not take control of a single instance of a HVAC unit, but they compromise the infrastructure that is used in order to take control of *all* connected devices. Such an attacker might be capable of causing all connected devices to shed their loads at the same time or to re-engage them at the same time. The grid does what the grid always does: It compensates. When a great load is shed and the line frequency rises, production is either brought down or additional load is connected automatically to bring the frequency to its desired operating point. Likewise, if there is a surge in consumption and frequency dips, production is brought up to compensate or other loads are shed.

The important thing to note here is that these regulation systems are not instantaneous. The connected systems have some inertia to it and so it is feasible in many cases to have regulation at a second-level granularity. The attacker has a strong position: They can simply monitor the line frequency and observe how fast compensatory measures kick in. This means, in term, they can essentially force resonance effects, doubling their attacking power: First, an attacker would shut down all devices under control. Then, when the attacker knows the regulation from the grid-side becomes effective and ramps production down, the attacker then simultaneously re-engages all devices under control. In a position where regulation is already going down and with the added inrush current of all new devices turning on at once, grid frequency can become dangerously low and, depending on the severity, a collapse might occur.

Security

What essentially happens during Demand Response negotiations is that devices need to communicate with the grid, where the grid has the privileged role (i.e., it can issue commands) and the devices have a subordinate role (i.e., they report state and other telemetry upstream). This is a classical server-client architecture that can be seen in much of today's cloud computing infrastructure as well. Additionally, we have the constraint that communication happens over an insecure transport, namely the Internet. If not protected, anyone on the path from client to server could eavesdrop or change data at will. To ensure confidentiality and data integrity – two major security goals of secure systems – we therefore need to wrap transmitted data in a layer of cryptographic protection. Developing such transport security is an incredibly complicated endeavor, but luckily, it is not necessary to reinvent the wheel: Protocols like TLS, the Transport Layer Security protocol [17, 6], are ubiquitously used today and have a high level of maturity.

While TLS does handle many technical aspects of protecting the transported data, it needs to be configured with proper security parameterization and it needs to know organizationally what peers are trustworthy to begin with. This is not something any technical protocol can answer, it is up to the implementer to decide. Firstly, parameterization of TLS is much more complicated than as it appears superficially. While specifications these days usually cover some aspects of TLS parameterization (e.g., [7]), many more intricate technical questions remain unanswered: there are many configuration settings such as key agreement methods (i.e., how are the session credentials negotiated between peers) – concretely, for example, which elliptic curves are deemed acceptable – or which TLS extension can or should be used (e.g., [2, 8]). All of these can have a profound impact on the security of any TLS connection even though superficially, all TLS connection look "cryptic". It is exceptionally difficult to decide for someone not intimately familiar with the ins and outs of TLS if a connection is configured in a way that is optimal in terms of security.

Organizational Security

On top of these technical challenges come organizational challenges: A technical protocol alone can ensure that the pure communication is secure, but how do systems decide if they are talking to the *correct, intended* peer? In the world of cryptography this is usually achieved by something called a *cryptographic binding* that combines an *identity* with a *technical measure* such as a public key. Typically, digital certificates achieve exactly this cryptographic binding and are used in the context of TLS as a kind of digital passport: It is a digitally signed, verifiable document that attests each participating party they are who they claim and includes the cryptographic means in which to securely communicate with them.

The issue that arises immediately with this scenario is: who are the entities which are even allowed to create these digital passports? In our analogy, for countries it is well-defined who may issue a passport and what the requirements for verification are before any individual is issued one. For the analogous *Certificate Authorities* (CAs), this is not so clear: Anyone can technically create own certificates with the press of a key, but in a system that deals with critical infrastructure, there need to be tight definitions on who is trusted. Note that this is always a question that arises when dealing with TLS and digital certificates, even if it is often invisible and overlooked: For example, when a user browses to a secure (`https`) web page, a TLS connection is negotiated under the hood and the peer website presents their certificate – the passport that links the domain name (e.g., `wikipedia.com`) to the key that should be used. The browser now needs to validate the passport and it checks this by verifying against a vast list of accepted issuing authorities. Many of these authorities have greatly differing levels of reliability and, in the past, this has led to spectacular attacks on such a Public Key Infrastructure (PKI) based system [20]. In consequence, such a breach usually leads to the browser vendor to respond by removing those CA certificates of which the operator has been proven they are unreliable [12, 15]. Coming back to Demand Response applications, the question that remains unanswered is: How can it be ensured that for these critical applications, only trustworthy CAs are used to issue certificates to devices and backends?

Revocation Capabilities and Contingency Planning

After the previous question has been answered, there is a direct follow-up that pertains to the operation of any PKI system: In case of a partial breach (i.e., a loss of either device or backend system's keys), how can it be ensured that the now untrustworthy certificates are revoked accordingly? To come back to the passport analogy: When somebody loses their passport, how can we ensure that no malicious person picks it up and uses it unauthorized? Technically, this is a well-studied topic, with Certificate Revocation Lists (CRL) [5] or the Online Certificate Status Protocol (OCSP) [18] being the technical measures for realization of said requirement. Efficient methods of using revocation information in TLS-secured connections are also available [16]. The questions posed are: What are the appropriate means of handling this technical issue and how are they solved organizationally – who has the authority to revoke certificates?

Thinking one step further, what happens in case a subset of used certificate has become invalid due to revocation? The system needs to be designed in a way that it can recover from such a threat and necessary contingency plans need to be in place before roll-out.

Future-Proof Cryptography

All these technical security goals which need to be fulfilled are practically implemented using cryptographic primitives. Such primitives include algorithms for encrypting bulk data, authenticating encrypted ciphertexts and key agreement protocols. With the continuing cryptanalysis of algorithms, new potential attacks are discovered. In particular, with some publications debating the technical feasibility of quantum computers, it might be possible that we do see practical implementations of a complete quantum computer system (where "complete" implies the ability of the computer to calculate Shor's Algorithm in polynomial time [19]) in the future. The theoretical framework has long been discussed by cryptographers around the world, only the practical implementation of the hardware is the last missing puzzle piece. In case such a computer became publicly available, all traditional asymmetric cryptography would essentially be instantly broken, and all symmetric cryptography be severely (but not necessarily devastatingly) reduced in strength. Research in the field of post-quantum cryptography (PQC) deals with exactly this eventuality and numerous schemes have been proposed that would remain secure even in the face of an available complete quantum computer. To this day, no scheme has emerged as clearly superior to the others and a lot of drawbacks are associated with PQC at this time, in particularly with regards to performance [14]. However, we do believe that for any future-proof system, it is a consideration and decision that should be made deliberately and revisited with the progress of cryptographic research over time.

Even for non-PQC cryptographic systems, there are many new cryptographic advancements over the last few years that should be considered and which not only can improve security of systems, but efficiency as well [1]. In contrast to PQC, these have already been widely adopted in our current Internet infrastructure and are already commonplace today [11, 9, 10].

Transport Security vs. Payload Security

When we are speaking of transport security, we mean that raw data is transported over a secure channel so that it is protected *in transit*. An inherent property of a system using transport encryption is that data, after it has passed the protected channel, comes out unprotected at the final destination. This is not a drawback or fault; it is simply a design implication of any transport security system. Imagine you have a phone connection that is secure in the sense that nobody can eavesdrop or interject on the phone line by tapping it: You can speak to a peer over that secure phone line and receive, for example, instructions on what to do. Now this action plan is in your head, but if you were to relay it to someone else, you could not possibly prove that you have received the instructions originally from a trusted source over a secure phone line.

To solve this issue, there is *payload security* that can be used and freely combined with transport security as well. In a payload security protocol, some or all messages are individually digitally signed. To get back to the analogy, this would be like short telegrams which all have individual signatures. As long as the telegram is passed on unaltered, any receiving party can verify the signature is intact and therefore, the telegram is authentic. In the context of Demand Response Capabilities it might make sense to augment transport security by payload security as well, to ensure that even if a compromise of a backend system occurs, an attacker is still unable to forge the necessary messages to make all devices do something that could be potentially harmful to the grid.

Software Maintenance

With all these technical and organizational issues, there is still an elephant in the room: All of this technical stack requires software to run. Typically, an operating system on which libraries operate, on top of which a cryptography layer protects the actual application data stream (e.g., TLS protecting HTTP communication), upon which the actual application is built. We therefore are discussing large software stacks, which are complex and non-trivial to put together; it is highly likely that the software contains defects that are undetected at the time of commissioning and which are only later revealed. In the context of Demand Response applications, such software defects, if they have security implications, can have catastrophic effects on the security of the overall system and need to be mitigated appropriately. Therefore, it is necessary to consider the maintenance and firmware update capabilities of devices in field. This would allow that, if during the lifetime of a device, critical defects become known, they can be remediated and quickly repaired before they become a widespread security issue.

Finally, considerations towards the backend system need to be made: How are decisions made regarding the actions end devices should take? Even when the security between backend and end devices is perfect, this is not of much use when the input that is the source of a control message can be arbitrarily generated or manipulated. A holistic approach seems sensible in this case that ensures the security of the whole ecosystem, not just the sum of its parts.

Testing

From the previous discussion, it should become clear that security must be an integral cornerstone of a Demand Response system. However, with security as a requirement come two major questions: Firstly, what is the appropriate level of security and assurance that is needed for the specific application? Secondly, how should devices be tested in a way that enables vendors to prove they have complied with all required security constraints?

To answer the first question, it should be debated on what the required level of assurance depends in the first place. For example, network accessibility of the devices could play a major role in this determination: A device that is exposed over a wireless interface poses arguably larger attack surface than another which connects only over wire-bound communication. Greater attack surface might justify more sophisticated protection mechanisms. Another consideration to make is the amount of risk that a device class poses. Unfortunately, this is not something that can be easily answered on a per-device basis: Imagine that the only criterion by which low to high assurance were the power delivery/consumption capabilities of said device. Then, a low-power 1 kW device might fall into the low assurance category. However, with attacks that *scale*, as we have discussed before, there can be thousands or millions of low-power devices in the field, which combined create a worthwhile target for an adversary. Therefore, the decision needs to be based on a combination of the aforementioned factors but need not rely on one by itself.

The second question we identified concerned the ability of vendors or manufacturers to reliably prove they have complied with security constraints and risk control measures. Penetration tests, i.e., attempts to identify vulnerabilities by testing a system, can vary greatly in their results depending on the actual capability of the analysts performing the test. Therefore, it seems desirable to define a security baseline and minimum assurance level against which devices shall be tested and to use third-party verification to ensure that these promises are actually kept in the respective implementations. It is not news anymore that IoT devices, where no restrictions or checks need to be performed, continuously make headlines whenever a large-scale attack is enabled by them [3]. More worryingly, however, is that even devices in which one would expect high assurance to be mandatory, such as implantable cardiac devices, are secured up to modern standards. Unfortunately, this is not the case in all instances [13].

Such missteps are usually not rooted in malice; in practice, it is more often than not that software systems have grown in complexity so tremendously while the surrounding development process remained unchanged. When we are speaking of embedded systems today, these are not a few hundred lines of hand-crafted assembly code like you would have seen 20 years ago – rather, we often see full embedded operating systems, typically with a Linux kernel, running a massive amount of userland code. This starts at systemd and libc, includes code for networking such as dhcpcd or avahi, covers a shell like bash, package manager such as apt and doesn't end with crypto libraries like OpenSSL or scripting languages such as Python. Naturally, managing dependencies between all of these components and tracking vulnerabilities even of transitively dependent components can be a challenging task. This is where a Secure Software Development Lifecycle (SSDLC) plays a major role in enabling developers to build secure products. For industrial control systems, standards like IEC62443-4-1 give a good example of what to look out for in order to achieve this goal [4]. In particular, 4-1 highlights the importance of adequate and appropriate staffing with people able to handle security capabilities, requires software assets are tracked in a traceable way, deals with handling of cryptographic material, how to manage field surveillance and proper patching discipline and much more. Having these cornerstones in place during development enables the manufacturer, once they are notified of a vulnerability in a particular sub-component, to answer the crucial questions: Is my product affected? If so, which releases did I build in the past which contained the vulnerability? Which of my patches contains the mitigation and what product version does that fix end up in? When the answers to all of these can quickly be determined, only then does it become easy for the vendor to give an accurate recommendation instead of being paralyzed by a vulnerability and not knowing how to proceed.

Conclusion

More integrated and decentralized grid infrastructure is something that we find will inevitably occur, and there are great benefits associated with it. As we have shown in this brief discussion, however, it is far from trivial to implement proper security and there needs to be alignment among stakeholders on what this means in the concrete context of Demand Response Capability systems. We recommend, in order to answer these questions, to have experts of their respective fields involved to determine cryptographic algorithms, protocols and security contingency plans which keep the ecosystem secure for the foreseeable future. An important question we also highlighted was the difficulty of validating that a device is actually secure according to the previously defined criteria. Independent testing of appliances and systems seems to be a promising way of delivering the assurance that the ecosystem requires in order to be kept secure. Furthermore, to consistently deliver this high software quality in a repeatable fashion, it is important to recognize the complexity of our software ecosystems today and adapt the surrounding processes appropriately. In particular, vendors having Secure Software Development Lifecycle in place is a necessary prerequisite for them to be able to react quickly in case of newly discovered vulnerabilities. It is our belief that with all these elements in place, a secure Demand Response infrastructure can be built and operated in a secure fashion.

References

[1] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 207–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[2] K. Bhargavan, A. Delignat-Lavaud, A. Pironti, A. Langley, and M. Ray. *Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension (RFC7627)*, September 2015.

[3] Ritesh Bhatia. The school kid who hacked over a million IoT devices. *Information Security Newspaper*, September 2019.

[4] International Electrotechnical Commission. *Security for industrial automation and control systems – Part 4-1: Secure product development lifecycle requirements Ed. 1.0*, January 2018.

[5] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (RFC5280)*, May 2008.

[6] Tim Dierks and Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2 (RFC5246)*, August 2008.

[7] Common Smart Inverter Profile Working Group. *IEEE 2030.5 Common California IOU Rule 21 Implementation Guide for Smart Inverters (Version 2.1)*, March 2018. `https://sunspec.org/wp-content/uploads/2018/04/CSIPImplementationGuidev2.103-15-2018.pdf`.

[8] Peter Gutmann. *Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) (RFC7366)*, September 2014.

[9] S. Josefsson and J. Schaad. *Algorithm Identifiers for Ed25519, Ed448, X25519, and X448 for Use in the Internet X.509 Public Key Infrastructure (RFC8410)*, August 2018.

[10] Y. Nir S. Josefsson. *Curve25519 and Curve448 for the Internet Key Exchange Protocol Version 2 (IKEv2) Key Agreement (RFC8031)*, December 2016.

[11] A. Langley, M. Hamburg, and S. Turner. *Elliptic Curves for Security (RFC7748)*, January 2016.

[12] Adam Langley. Maintaining digital certificate security. March 2015. `https://security.googleblog.com/2015/03/maintaining-digital-certificate-security.html`.

[13] Selena Larson. FDA confirms that St. Jude's cardiac devices can be hacked. *CNN Business*, January 2017.

[14] Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, and Audun Jøsang. The impact of quantum computing on present cryptography. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 9(3), 2018.

[15] Devon O'Brien, Ryan Sleevi, and Emily Stark. Distrust of the Symantec PKI: Immediate action needed by site operators. March 2018. `https://security.googleblog.com/2018/03/distrust-of-symantec-pki-immediate.html`.

[16] Y. Pettersen. *The Transport Layer Security (TLS) Multiple Certificate Status Request Extension (RFC6961)*, June 2013.

[17] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3 (RFC8446)*, August 2018.

[18] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP (RFC6960)*, June 2013.

[19] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, November 1994.

[20] Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. Cryptology ePrint Archive, Report 2009/111, December 2009. `https://eprint.iacr.org/2009/111`.