Analyzing Cross-platform Attacks: towards a Three-actor Approach

Antonio Acien, Ana Nieto and Javier Lopez

Network, Information and Computer Security (NICS) Lab
Lenguajes y Ciencias de la Computación
Universidad de Málaga, Spain
Email:{acien,nieto,jlm}@lcc.uma.es

***Abstract*—In the current telecommunications landscape, different devices, systems and platforms are constantly communicating with each other. This heterogeneous environment creates the perfect situation for attacks to pass from one platform to another. This is a particularly worrying scenario, because of the new technologies being used (such as network slicing in 5G), the increasing importance of connected devices in our lives (IoT), and the unpredictable consequences that an attack of this type could have. The current approaches in attack analysis do not take into account these sitations, and the attacker/victim paradigm usually followed may fall short when dealing with these attacks. Thus, in this paper, an architecture for the analysis of cross-platform attacks will be presented, aiming to help understand better this kind of threats and offering solutions to mitigate and track them.**

***Index Terms*—Cross-platform, architecture, attack, security.**

I. Introduction

Cross-platform attacks (those which affect several platforms and services) are hard to detect and track, since the vast majority of security measures are limited to the security of one platform. In this paper, we will be focusing on cross-platform attacks in which the operation of the intermediary transmissors is not directly affected, making the attacks harder to detect before they reach their target. In an environment where different platforms, technologies and protocols are more and more interconnected between themselves, these kind of attacks pose a serious risk.

Throughout this paper, it will be explained how a certain platform can be compromised from a really different one, using intermediary elements in the communication. Since, to our understanding, there is no other architecture for analyzing these threats that allows to understand their progression in real time, one is proposed in this paper. The proposed architecture aims to help with the detection, prevention and mitigation of such attacks.

It must be noted that intercommunication between different platforms, or layers of the same platform (such as network slices in 5G), is becoming a common scenario in modern communications. It is not strange to have an smartphone connecting to a computer, which in turn connects to several other devices (routers, USB gadgets, or other smartphones), or even to a car which has a connection to a wireless network. A similar landscape is also found in virtualized environments (where the guest connects to the host) or in systems composed of several embedded systems which communicate with each other. The idea of an attack infecting one of these platforms, and spreading to the ones communicated with it is a reality that has not been just seen in theoretical proofs of concept or controlled environments, but in running systems in the real world and regular user devices (section II). With the surfacing of new technologies promoting these interactions (5G, mmWave, SDN...), it is necessary to establish tools that make possible an analysis in order to understand this context.

The paper is structured as follows. In section II, an study on the state of the art regarding cross-platform attacks is presented, and after that, the proposed architecture, called BTV (Bearer, Transmitter Victim) will be shown in section III. It will be formalized in section IV. The section V describes how the BTV architecture would be applied in a simplified use case. Last, a discussion on the results and some conclussions drawn from the work are carried out in sections VI and VII, respectively.

II. State of the art analysis

The definition of cross-platform attacks is relatively wide, and it can refer to several meanings. Ranging from attacks that, with the same code, can affect different platforms (for example, taking advantage of the multi-platform feature of Java[1]), to those that modify their code depending on the system they are targeting (metamorphic attacks) [2]. Moreover, some works on the cross-platform concept opt for focusing on permissions and policies for specific scenarios, or procedures, methods and libraries that have counterparts in different operating systems or architectures [3].

Most of the contributions in this area are related to networks in general: packet transmission, authentication and acccess control, among other topics [4]. Also, the networks taken into account have been specific technologies, such as 3G [5][6]. In addition, some specific cases of vehicular networks are studied, where cars are platforms which, in turn, integrate other platforms. For example, in [7], a CD is used to cause disruption in other parts of the car system, such as the engine or the steering, and in [8], a paired Bluetooth device sends messages to other parts of the vehicle, such as multimedia nodes, or the parts which manage external communications.

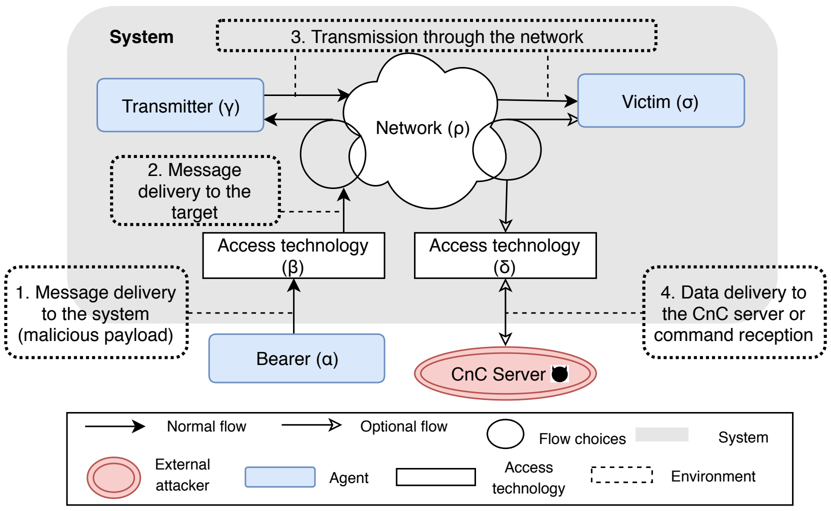

Fig. 1. BTV architecture diagram

There have been also cases in virtualization environments, where the host machine was supposed to be a platform isolated from the virtualized one. Nontheless, it is known that there are attacks designed to break out from this envirmoment, surpassing the guest-host barrier, and allowing to write in memory positions restricted to the host machine from the guest one. These attacks go further than just proofs of concept [9], since there are exploits commercially available to carry out the attack in a real environment [10].

The Internet of Things (IoT) is not an exception either. For example, the Windows version of Mirai attacks to computers with this operating system, but once they are reached, it scans the local network searching for vulnerable IoT devices, which can be turned into part of a botnet, in order to carry out distributed denial of service (DDoS) attacks towards other infrastructures [11].

Furthermore, some attacks which are transmitted from an smartphone to a computer and vice versa have been documented. In the former, normally they are transmitted when the mobile device is connected throguh USB, turning the computer microphone into a wiretapping device and sending the recorded audio to an external server, spying conversations [12]. Following the opposite flow, some attacks spread from a computer to an Android device, deleting some installed apps and replacing them with identical versions which send the identification data entered to a rogue server [13]. This is specially delicate when these apps have sensible info (such as mobile banking apps). Also worth discussing, there is XcodeGhost, which is a modified version of the integrated development environment (IDE) for iOS which infects the applications programmed with it, and spreads this malware to the devices where they are installed [14].

It must be pointed out that, despite the fact that cross-platform attacks have become more worrying and common in recent times (mainly motivated by advanced persistent threats), they are nothing new, although the terminology and approach given in this work are. The spread of attacks through multiple platforms is something that has been happening since several communication generations back, and it has been seen that although improvements are made in security, if there are no solutions fitting the context changes, it will be very hard to stop the spread of attacks in bearer elements. Any security model should consider this type of attacks and face them from a global perspective. With the development of new platforms and technologies, there is also a rise of malware targeted specifically to them [15], which lays bare the need to face this problem. Therefore, defining a strategy and measures for these attacks is prioritary.

This paper will provide a general approach to this problem, with the goal of offering solutions that allow to answer to the attacks, regardless of the platform.

III. Architecture for cross-platform attacks analysis

In this section, the BTV architecture (Bearer, Transmitter, Victim) is presented. It aims to cover the different scenarios where cross-platform attacks take place in a simple and general fashion, with enough flexibility to adapt to each of the different situations and attack flows. The main idea behind the concept of BTV is proposing a simple schema which consists of an agent which holds the attack, known as *bearer*. This attack is targeted to a victim, which is not infected directly. Instead, the attack is passed through another agent, called *transmitter*, which holds the attack, but has no alteration in its operation (or at least, not obvious enough to make the attack detectable to the end-user). Last, the transmitter will unconsciously send the attack to the victim, who suffers the consequences of the attack. After explaining this idea in a graphic and detailed manner (section III-A), it will be explained how the architecture would be used to identify cross-platform attacks (section III-B).

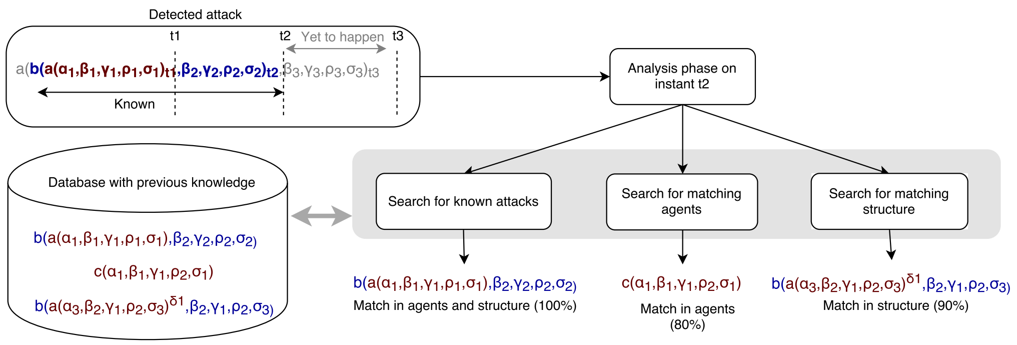
Detected attack
t1
t2
Yet to happen
t3
a(b(a(α1,β1,γ1,ρ1,σ1)t1,β2,γ2,ρ2,σ2)t2,β3,γ3,ρ3,σ3)t3
Known
Analysis phase on instant t2
Database with previous knowledge
b(a(α1,β1,γ1,ρ1,σ1),β2,γ2,ρ2,σ2)
c(α1,β1,γ1,ρ2,σ1)
b(a(α3,β2,γ1,ρ2,σ3)δ1,β2,γ1,ρ2,σ3)
Search for known attacks
Search for matching agents
Search for matching structure
b(a(α1,β1,γ1,ρ1,σ1),β2,γ2,ρ2,σ2)
Match in agents and structure (100%)
c(α1,β1,γ1,ρ2,σ1)
Match in agents (80%)
b(a(α3,β2,γ1,ρ2,σ3)δ1,β2,γ1,ρ2,σ3)
Match in structure (90%)

Fig. 2. Implementation of a threat detection system based on the BTV architecture

A. Architectural components

The classic approach taken when analyzing attacks (section II) is a simplified scenario with two actors. In such situation, both attacker and victim are well-defined agents: the attacker makes an effort to spread the infection to the victim, who is compromised and harmed. However, this standpoint may fall short in cross-platform attacks, which involve more systems with different roles. Systems are prepared to detect malware targeted towards themselves, but not to others. When a situation where a device is infected but its operation does not change arises, detecting the attack is not easy at all. The role of this unconscious transmitter is essential in many cases, since it allows attacks to affect devices that could not be accessed otherwise.

The BTV architecture (Fig. 1) allows to study the possibility of an attack originated in a platform spreading to different ones, applying from new scenarios, such as 5G and SDN, or vehicles with data connection, to daily situations such as USB or Bluetooth connections between smartphones and computers.

In the schema there are three main actors, which are now described:

1) **Bearer:** this participant in the architecture is the one who starts the propagation of the attack. It can be a malicious attacker with the goal of causing harm, or an unconscious user who believes to be operating normally.
2) **Transmitter:** the agent who recieves the attack, but is not directly affected by it.
3) **Victim:** the target system of the attack. Usually, it communicates with a CnC (Command and Control) server to receive instructions or send data, although it is not always the case, since it could be the victim of an attack without need of external communication.

As it can be seen in Fig. 1, there are more parts involved. One of them is the access technology, which, depending on the type of attack, will vary. If the compromised agent is an internet server with malware, it could access the transmitter simply through an average router, whereas if it is a vehicle, this access will be through a gateway for external communications or an ECU (Electronic Control unit).

Also there is a network expressing the connection between the transmitter and the affected agent. This can be via WiFi, USB, Bluetooth, CAN, or any other protocol. Messages can be sent through it to each of the agents, or directly through the access technology.

It must be pointed out that, while in some scenarios agents are independent devices with momentary connections (such as a smartphone and a computer), in others, they are part of a complete indivisible system, where they share resources or messages, such as the parts of an automobile, or a virtual machine and its host. In addition to those systems, environments can be defined, which represent scenarios where usually these agents interact.

B. Implementation

Having the possibility of analyzing the attacks in a systematic way through this architecture, makes way for the development of applications using it, that through its implementation, help preventing, detecting and mitigating these threats.

Once the known cross-platform are analyzed (those in bibliography, or those detected in production environments), their spreading patterns can be used as previous knowledge. If they are included in a database, comparing their participants or structure with the detected threats can help predict their evolution and prevent their propagation, or taking the right measures to mitigate them.

The comparison with known attacks can be carried out in several ways. The attacks can be compared through their structure, thus, checking if the involved agents follow a certain pattern, and predicting if the attack will spread to another platform. It can also be checked if the involved agents match those of some known case, identifying previously known attacks, or variations of them.

When these checks are made, trying to match the information about attacks in the database, a criteria for similarity must be established. This will help deciding which measures should be taken, because if the similtarity to a certain threat is high, it can be used as a reference to deal with the detected one.

This similarity rule can be based on different variables. One of the comparisons to be made is a structural one: if the attacks follows the same structure as a known one (although through different agents), it can be predicted if it is going to spread and how. However, if the structure is different but the agents match, it can be predicted which agents could be affected if the attack keeps spreading with such structure. Of course, both criteria can be met, matching a known attack in all of its variables. These cases are illustrated in Fig. 2. It must be emphasized that the arrows indicating the connection between the detected attacks and the observed ones in the database, would change as data about the threat is gathered. This is expressed through the time bar t_n.

As noted, one of the biggest difficulties to overcome in cross-platform attacks is their detection, because the platforms are usually prepared to detect and mitigate attacks targeted specifically towards them. This task is hindered if the attack is harmless for them, and they just collaborate in its spreading. Through this previous knowledge, expressed through the architecture, this problem could be faced, since when an attack is detected and compared with the data, the possibilities of it spreading are analyzed and taken into account.

IV. Formalization

With the aim of helping define the architecture, it has been decided to formalize its structure with a simple but versatile language, in a way that it can be instantiated in any case necessary. Having a way to express different scenarios in an unified manner makes possible for machines to understand it, helping its automatic processing and thus, having a more direct application. This will be helpful when carrying out the detection detailed in section IV, as the commented comparisons can be made systematically, comparing variables or expressions.

A. Attack vector

The general expression which summarizes the architecture is the following one:

$$V_a = f\,x^{\delta}_{t_{\mathbb{N}}}\,,\; x = (\alpha, \beta, \gamma, \rho, \sigma) \tag{1}$$

Through this language, which makes use of different variables, every scenario considered by the BTV architecture can be expressed. This expression will describe the *attack vectors* (V_a) of the architecture. Next, the sets, variables and functions which give meaning to this formula are thoroughly described. The table I sums up all the formalization.

B. Sets

Two sets are defined: A and T, expressing the possible agents and technologies, respectively. These are sets consisting of character strings. Since it is not possible to include all the possibilities in these sets, they will be instantiated in each use case, including the necessary members.

Example:

$$A = \{Server, Smartphone, CANNode...\} \tag{2}$$
$$T = \{USB, WiFi, Bluetooth, ZigBee, CAN...\} \tag{3}$$

C. BTV architecture variables

Several variables are used, which can be seen graphically in Fig. 1. In order to symbolize the bearer, $alpha$ is used. As seen in exprs. (4), it can be in turn another attack vector, providing recursion. The access technologies are expressed through $beta$, $gamma$ is used for the transmitter, rho stands for the network, and $sigma$ for the victim. The time instant in which the attack takes place is symbolized through $epsilon$, which can vary as more information is known. Last, $delta$ represents the technology used to communicate with the CnC server, if there is any, thus being optional.

$$\alpha \in \{V_a, A\} \tag{4}$$
$$\beta, \rho, \delta \in T \tag{5}$$
$$\gamma, \sigma \in A \tag{6}$$
$$\epsilon \in t_{\mathbb{N}} \tag{7}$$

D. Flow functions

In order to represent the flow of the attack in the vector, each one of the parts through which the attacks passes by can be indicated sequentially. However, this would make attack vectors heterogeneous, with a different amount of members each time, depending on the situation, and some of them could be repeated. To avoid these issues, flow functions represent the way the attack takes through the agents it affects. The function is expressed generally as f, but will be instantiated as specific flow functions which describe detailed situations (seen in Fig. 3): a, b, c or d.

$$f\,x = \begin{cases} a\,x & \text{si}\,\alpha \to \beta \to \gamma \to \rho \to \sigma \\ b\,x & \text{si}\,\alpha \to \beta \to \boldsymbol{\rho} \to \gamma \to \rho \to \sigma \\ c\,x & \text{si}\,\alpha \to \beta \to \gamma \to \rho \to \sigma \to \boldsymbol{\rho} \\ d\,x & \text{si}\,\alpha \to \beta \to \boldsymbol{\rho} \to \gamma \to \rho \to \sigma \to \boldsymbol{\rho} \end{cases} \tag{8}$$

It must be noted that functions a and c are very similar, and the only distinction present is that in case of an attack spreading (to another environment, or CnC server), the passing is made through the network. Therefore, having an attack *mutate* from a function to another in its operation is possible. A similar situation arises between functions b and d.

E. Delimiters

Aside from previously presented variables, the symbols [ ] and { } are also included. These are used to demarcate the extension of a system and an environment, respectively. Having agents comprised between the limits of a system means that they are indivisible and they have a joint functioning. Environments represent sets of devices which normally interact between them in order to carry out certain functions, but they can work independently. There are certain rules to follow when using these symbols:

1) Both an environment and a system can exist one without the other. Nonetheless, an environment cannot be

TABLE I
BTV ARCHITECTURE FORMALIZATION

Sets	**Delimiters**
A: Set of possible agents taking part in the scenarios represented by the architecture T: Set of the possible technologies taking part in the scenarios represented by the architecture	$[]$: System delimiters $\{\}$: Environment delimiters Example: $f(\alpha,\{\beta,[\gamma,\rho]\},\sigma)$
Attack vector	**BTV Components**
$V_a = f\,x\,^{\delta}_{t_{\mathbb{N}}}$ **Flow function** $f\,x = \begin{cases} a\,x & \text{si}\,\alpha \to \beta \to \gamma \to \rho \to \sigma \\ b\,x & \text{si}\,\alpha \to \beta \to \boldsymbol{\rho} \to \gamma \to \rho \to \sigma \\ c\,x & \text{si}\,\alpha \to \beta \to \gamma \to \rho \to \sigma \to \boldsymbol{\rho} \\ d\,x & \text{si}\,\alpha \to \beta \to \boldsymbol{\rho} \to \gamma \to \rho \to \sigma \to \boldsymbol{\rho} \end{cases}$	$\alpha \in \{V_a, A\}$ $\beta,\rho,\delta \in T$ $\gamma,\sigma \in A$ $\epsilon \in t_{\mathbb{N}}$ α: Attack bearer β: Transmitter access technology ρ: Network communicating the transmitter with the victim σ: Attack victim δ: Access technology between the victim and the CnC server (if any) $t_{\mathbb{N}}$: Time instant

comprised inside of a system. A system can indeed be comprised inside an environment.

2) In the case of a system which starts inside an environment, it must also end inside of it.
3) Both system and environment delimiters act on variables $\alpha, \beta, \gamma, \rho$ y ϵ, and cannot be applied to others.

For example, if there is an enviroment containing the access technology, the transmitter and the network, and a system spanning these last two, it will be expressed as $f(\alpha,\{\beta,[\gamma,\rho]\},\sigma)$.

Once the context to be analyzed is defined using this formalization, implementing a mitiagtion system based on the BTV architecture is easier, since all the attacks can be expressed through formulas with a common syntax.

V. USE CASE

In this section, a use case making use of several technologies will be studied. This way, it will be observed how the BTV architecture can be applied to different scenarios and protocols, showing its utility in real practical situations. Although in the use case, specific technologies and devices will be mentioned, it must be pointed out that there would be infinite variations. Likewise, the use case will be formalized using the functions and variables seen in the previous section.

In the first phase of the attack, an internet server hosting malware infects a computer through an exploit, taking advantage of a vulnerability in the web browser, in order to install an executable file with malicious code (steps 1 and 2). The computer user is unaware of the malware being installed on the machine, because it does not affect its regular operation, but it is in the background, monitoring the USB connections until it finds an Android device (step 3). When this happens, the malware will look for an app which connects to a vehicle via Bluetooth, uninstalling it (step 4), and replacing it with a rogue version (identical to the eyes of the user) that connects with the CnC server of the attacker (step 5) and has malicious purposes. There have been cases of malware following this operation patter with other types of applications, such as banking apps [13].

Thus, the bearer in this phase would be the external server, while the computer and the Android device would have the roles of transmitter and victim, respectively. In turn, the smartphone could be a bearer if the attack keeps spreading.

In order to express this part of the use case through the formalization presented in section IV, the following abbreviations will be used for the sake of brevity in the formulas. These have been chosen in a completely arbitrary manner, and they can be extended as much as wanted, but there must be coherency, using always the same ones inside of an implementation if one is carried out.

The application of the formula which formalizes the attack is quite straightforward, given that it is only needed to replace the variables and functions with the ones used. In order to keep it simple, the abbreviations in table II will be used. In this case, the result would be the following:

$$V_a = f(\alpha_1, \beta_1, \gamma_1, \rho_1, \sigma_1)^{\delta}_{\epsilon} \tag{9}$$

$$V_{aCU} = a(SRV, NVG, PC, USB, AND)^{DAT}_{t_1} \tag{10}$$

TABLE II
TABLE OF ABBREVIATIONS FOR THE USE CASE

Acronym	**Meaning**
SRV	Server
PC	Computer
AND	Android device
BT	Bluetooth
OBC	On-board computer
GW	Gateway
GW-BT	Gateway (Bluetooth connection)
LC	Light control
NVG	Browser
DAT	Data connection
NODE	Node
ENG	Engine

Since there is no CnC server in this scenario, it is not included in the previous expression. It can be seen that, since this scenario is analyzed statically, and not through time, the time instant is not included as a subindex.

Once the smartphone has been infected as seen, the installed app can monitor Bluetooth connections with vehicles. This

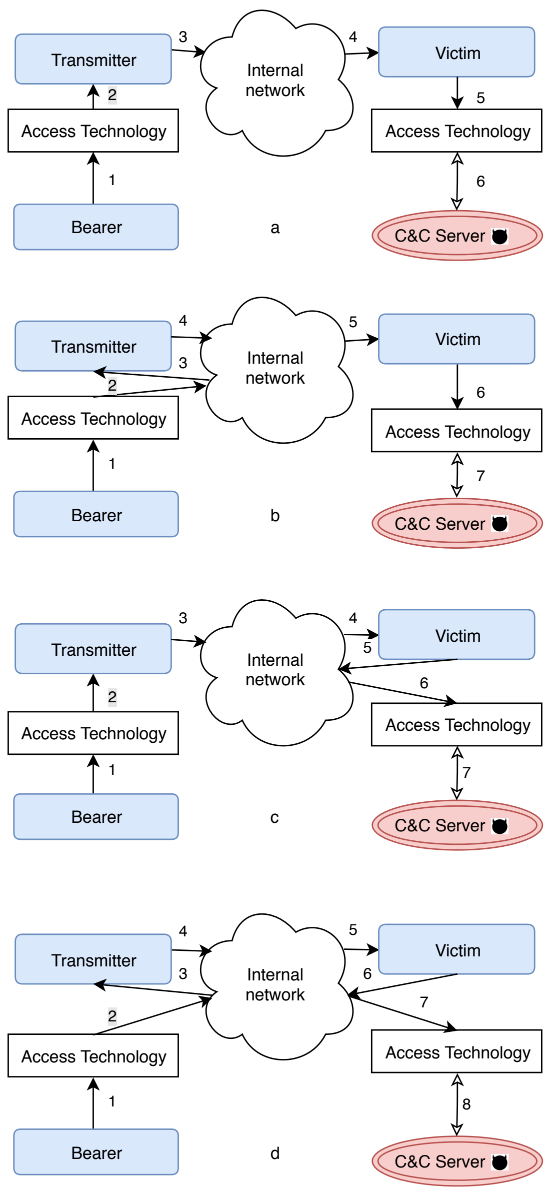

Fig. 3. Possible attack flows which the different functions represent

means that when pairing with a vehicle, the app can try to compromise its operation. The connections inside cars follow the CAN protocol (*Controller Area Netowrk*), which regulates the communications through a bus which the different nodes are connected to. These nodes can be as diverse as lights control or even the engine. Due to the design of this protocol, it is very easy to carry out denial of service attack from any of the nodes, sending a massive amount of messages, thus flooding the bus.

There is another kind of more sophisticated attacks, which can be targeted to some of the nodes comprising the internal network of the automobile, originating from any of them. These attacks, however, require a deep understanding of the firmware in the nodes, which depends on the exact model of the vehicle, and they are overall more complicated.

Both in simpler and more complicated attacks, the scenario would consist of an Android smartphone connecting to an automobile system through Bluetooth. In modern vehicles, these communications are usually made through a gateway meant for external connections. Once the device is paired, it could start sending a massive amount of messages through said gateway (step 6), blocking the communications through the bus for other nodes, making a denial of service. Although some modern vehicles define different levels of communications, and implement a bus for each of them, separating critical communications from more secondary ones through firewalls, there have been attacks where these security mechanisms have been bypassed [8][16]. Another kind of attack could have the smartphone sending a message to the on-board computer to have it redirected to another node, which would be the victim (step 7). This message would contain a malicious payload affecting the victim, but not the transmitter, due to its encoding. This node could be any of the network (windows, speedometer, fuel indicator...), but for the sake of simplicity, it will be instantiated to the light control in this case (step 8).

As previously seen, the Android device is no longer the victim of the attack, but the bearer, since it can spread to other platforms. In this case, the transmitter would be the external communication gateway, which is accessed through Bluetooth, the access technology. Once the transmitter is infected, it will continue operating as usual (handling the messages of the devices connected to it), but the rest of the nodes of the vehicle, which are the victims accessed through the CAN bus, will not be able to function properly, since the smartphone will have flooded the bus with packets, causing a denial of service.

Considering that the smartphone, which is a bearer, was in turn a victim, this scenario can be described as a recursion. Moreover, since the messages are sent through the CAN bus from the access technology to the on-board computer, the function changes in this recursion. The resulting expression is the following one:

$$V_a = f(f(\alpha_1, \beta_1, \gamma_1, \rho_1, \sigma_1)^{\delta}, \beta_2, \gamma_2, \rho_2, \sigma_2)_{\epsilon} \tag{11}$$

$$\begin{aligned} V_{aCU} = b(a(&SRV, NVG, PC, USB, AND)^{DAT}, \\ &GW - BT, OBC, CAN, LC))_{t_2} \end{aligned} \tag{12}$$

This attack takes advantage of the simplicity that the CAN protocol offers, since it has been designed to work on micro-controllers with very little processing power, and it is focused on real-time communications rather than security [17]. Thus, the mechanisms to avoid packet injection or denial of service attack would worsen the response time of the system, given that they require complex operations.

Now that the formalization of the attack observed in the use case has been made, it can be compared to the implementation detailed in section III-B with registered attacks. These attacks are both real threats registered in malware databases[13] and academic proofs of concept [18].

It is seen how the attack is similar to the trojan TROJ˙DROIDPAK.A [13], since they share the same agents (a personal computer infected from an external connection,

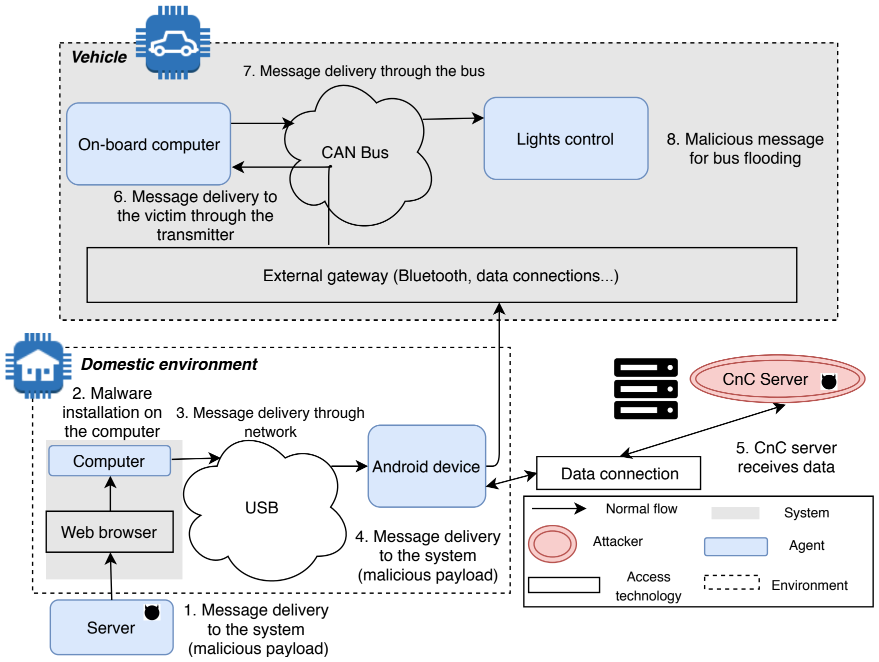
Vehicle
7. Message delivery through the bus
On-board computer
CAN Bus
Lights control
8. Malicious message for bus flooding
6. Message delivery to the victim through the transmitter
External gateway (Bluetooth, data connections...)
Domestic environment
2. Malware installation on the computer
3. Message delivery through network
Computer
USB
Android device
Web browser
4. Message delivery to the system (malicious payload)
Server
1. Message delivery to the system (malicious payload)
CnC Server
Data connection
5. CnC server receives data
Normal flow
System
Attacker
Agent
Access technology
Environment

Fig. 4. Application of the BTV architecture in a use case

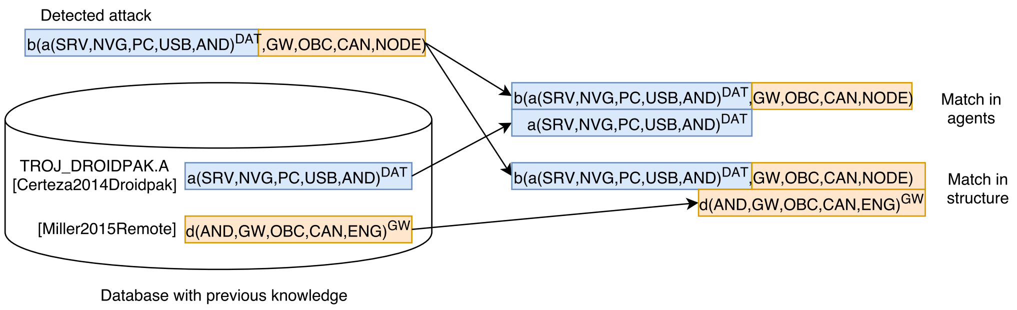
Detected attack
b(a(SRV,NVG,PC,USB,AND)^DAT,GW,OBC,CAN,NODE)
TROJ_DROIDPAK.A [Certeza2014Droidpak]
a(SRV,NVG,PC,USB,AND)^DAT
[Miller2015Remote]
d(AND,GW,OBC,CAN,ENG)^GW
Database with previous knowledge
b(a(SRV,NVG,PC,USB,AND)^DAT,GW,OBC,CAN,NODE)
a(SRV,NVG,PC,USB,AND)^DAT
Match in agents
b(a(SRV,NVG,PC,USB,AND)^DAT,GW,OBC,CAN,NODE)
d(AND,GW,OBC,CAN,ENG)^GW
Match in structure

Fig. 5. Comparison of the use case with known attacks, using the proposed formalization

transmitting to a smartphone connected via USB) and with the automobile attack described by Miller and Valasek [18], because the structure is quite similar (packets are sent through an external connection in order to harm the operation of a vehicle node). Thanks to these similarities, measures can be taking to stop propagation, or mitigating the effects of the attack, since these are detailed in the papers and the malware databases [19], such as modifying Windows registers of closing the external connections to vehicles.

As it has been shown, the BTV architecture is extendible to several situations, different platforms and protocols, and versatile enough to adapt to more complex attacks.

VI. Discussion

There are several details worth commenting regarding cross-platform attacks and the approach taken when developing the BTV architecture. On the one hand, it would be interesting to consider how feasible it is to carry an attack with the importance of those commented in the paper on a real system. For example, for more sophisticated attacks, developing a rogue firmware for a vehicle is not a trivial task by any means. Even though in some proofs of concept, changing a few lines of code has changed the behavior of the car in a crucial manner [8], an attacker would need access to the original firmware, which is not easy to obtain, since reverse engineering is an ardous and long task. Nonetheless, the worry about this is growing, as recent news and studies show [20]. Moreover, regarding the attacks in virtualization platforms, it must not be forgotten that certain settings or versions are needed for them to work, and these are not easily found in real systems.

On the other hand, even taking into account the peculiarities of these situations and how specific the scenarios needed for carrying out the attacks can be some times, it does not mean that they are not feasible. Each part of the use cases shown is based on a real attack, thus there is a real risk of them growing bigger and having fatal consequences for potential victims. The BTV architecture aims covering both these scenarios as well as the new ones arising. In order to make this possible, a comparison rule is needed, both variable and structure-wise. A similarity parameter to each of the known attacks can be established based on these operations, helping know which attack matches the most, in order to follow the necessary security measures.

The method to follow in order to search for cross-platform attacks in attack databases or bibliography is also an issue to take into account, since it is crucial to develop a strong database of previous knowledge.

The presence of new telecommunication landscapes also brings new security challenges. For example, with network slicing in 5G, devices are grouped in isolated slices, which are adapted to their needs (speed, priority...). If an attack originating in one slice, could compromise the slice manager and spread to another one, a critical infrastructure could be infected from a simple user-end terminal. The BTV architecture can also be used to analyze newer scenarios, such as this one.

Last, this approach would be very beneficial for the fifth generation of cellular networks (5G) for a couple of reasons. First, the great synergy of software technologies cooperating to offer service, using a myriad of different platforms and integrating the final user (and the devices used) more than ever before as part of the ecosystem. Second, for operational and deploying reasons. This architecture and its implementation require a great resource capacity in order to function correctly. The processing of attack vectors in real time needs advanced handling and classifying a huge amount of data in an efficient manner. These service could be provided by the 5G infrastructure from the core.

VII. Conclussions and future work

Cross-platform attacks are a reality spreading more and more, and as the communication infrastructures improve, the propagation possibilities do so too. In this paper, an architecture able to express the variations of cross-platform attacks is defined and formalized, providing a specialized approach to each case, but sharing a common language to convey the attack vectors and making their comparison easier. The goal is to help the prevention and mitigation of these threats, allowing the intermediate components to predict if something happening could potentially affect other systems it communicates with.

Regarding future work and improvements, the direct and practical application of this work could be tested using the BTV architecture to solve threats in simulated environments through tools such as ns3 or OMNET++, offering solutions to attacks such as denials of service, on how to stop their spreading, and obtaining data to help the systems implement security measures in highly complex environments, like 5G.

Acknowledgment

This work has been financed by the Ministerio de Economía y Competitividad through the projects IoTest (TIN2015-72634-EXP) and SMOG (TIN2016-79095-C2-1-R). The second author has been financed by INCIBE through the program of aids for the excellence in advanced cybersecurity research teams.

References

[1] M. Lindorfer, M. Neumayr, J. Caballero, and C. Platzer, “Poster: Cross-platform malware: write once, infect everywhere,” in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 1425–1428.

[2] Eugene, “Architecture spanning shellcode,” 2000.

[3] K. Chen, X. Wang, Y. Chen, P. Wang, Y. Lee, X. Wang, B. Ma, A. Wang, Y. Zhang, and W. Zou, “Following devil’s footprints: Cross-platform analysis of potentially harmful libraries on android and ios,” in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 357–376.

[4] C. R. Reeves Jr, “Cross platform network authentication and authorization model,” Feb. 13 2007, uS Patent 7,178,163.

[5] W. Jia, D. Bin, and L. Liao, “Architecture of secure cross-platform and network communications,” in *Proceedings of the 2nd international conference on Ubiquitous information management and communication*. ACM, 2008, pp. 321–328.

[6] K. Kotapati, P. Liu, Y. Sun, and T. LaPorta, “A taxonomy of cyber attacks on 3g networks,” *Intelligence and Security Informatics*, pp. 129–138, 2005.

[7] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno *et al.*, “Comprehensive experimental analyses of automotive attack surfaces.” in *USENIX Security Symposium*. San Francisco, 2011.

[8] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, “Experimental security analysis of a modern automobile,” in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[9] N. Elhage, “Virtunoid: Breaking out of kvm,” *Black Hat USA*, 2011.

[10] K. Kortchinsky, “Cloudburst: A vmware guest to host escape story,” *Black Hat USA*, 2009.

[11] M. Mimoso, “Windows botnet spreading mirai variant,” 2017.

[12] “Android.ciaco,” 2013.

[13] R. Certeza, “Cross-platform mobile threats: A multi-pronged attack,” 2014.

[14] X. Gui, J. Liu, M. Chi, C. Li, and Z. Lei, “Analysis of malware application based on massive network traffic,” *China Communications*, vol. 13, no. 8, pp. 209–221, 2016.

[15] M. Sargent, “Malware trends: The rise of cross-platform malware,” 2012.

[16] L. Mearian, “Firewalls can’t protect today’s connected cars,” 2015.

[17] R. Bosch *et al.*, “Can specification version 2.0,” *Rober Bousch GmbH, Postfach*, vol. 300240, p. 72, 1991.

[18] C. Miller and C. Valasek, “Remote exploitation of an unaltered passenger vehicle,” *Black Hat USA*, vol. 2015, 2015.

[19] G. R. Joi, “Troj˙droidpak.a,” 2014.

[20] F. Maggi, “The crisis of connected cars: When vulnerabilites affect the can standard,” 2017.