Automotive Exploitation Sandbox: A Hands-on Educational Introduction to Embedded Device Exploitation

Nathaniel Boggs
n@redballoonsecurity.com
Red Balloon Security

Ang Cui
a@redballoonsecurity.com
Red Balloon Security

Jatin Kataria
j@redballoonsecurity.com
Red Balloon Security

Philippe Laulheret
philippe@redballoonsecurity.com
Red Balloon Security

Abstract

Automative managers, domain experts, and interested nontechnical personnel who have traditionally not needed to be aware of security issues, now face the at times daunting task of familiarizing themselves with the whole field of security knowledge. The goal of the Automotive Exploitation Sandbox is to educate stakeholders who may not have any security background and provide hands-on exposure to automotive attack chains with real hardware. In this paper and the accompanying presentation, we describe the goals and creation of the Automotive Exploitation Sandbox along with a live demonstration. The Automotive Exploitation Sandbox will be hosted and open for free public usage at the conclusion of the presentation at https://sandbox.redballoonsecurity.com.

1 Goals

The goals of the Automotive Exploitation Sandbox are twofold:

- To educate all stakeholders about what typical automotive attack chains look like.
- To provide hands-on experience with real hardware.

Providing a hands-on experience requires realistic exploits. At the very least, the exploits used should be real code exploiting a synthetic vulnerability implanted into the device used for the sandbox. To provide further realism, known existing vulnerabilities could be used. While going into the full technical instructions sufficient to have users write their own exploit implementations is out of scope, part of the educational experience provided by the Automotive Exploitation Sandbox will be overviews of how such vulnerabilities are found and exploits are made.

In order to be accessible to all stakeholders including management and automotive engineers, the Automotive Exploitation Sandbox must be:

- Free and publicly accessible.
- Clearly described with instructions targeted towards people without any particular technical or security background.
- Remotely accessible as even the most inexpensive hardware platforms are cost prohibitive for wide distribution.

2 Design

The design of the Automotive Exploitation Sandbox centers around the requirements for remote accessibility as well as to use physical hardware to achieve realism. The first step of the design process was choosing a target device. We choose the SABRE lite board. The SABRE lite is a relatively low cost development board with a quad core ARM processor. Importantly, it supports QNX, a micro kernel operating system common in automotive devices.

Using QNX as the operating system for the Automotive Exploitation Sandbox provides additional realism over using a common consumer focused Linux operating system. With an operating system selected, the next software required is a network server. Lighttpd provides a simple light weight web server. In practice, many embedded devices host a loosely secured web server. In the future, other network facing services can be used to demonstrate a larger variety of services. A command injection vulnerability is intentionally injected into the Lighttpd server to provide access to a user privilege level remote shell.

With the device and software chosen, the remaining Automotive Exploitation Sandbox design focuses on the infrastructure required to cleanly reset the devices, securely host the sandbox, and host the exploitation instructions. Figure 1 provides an overview of the physical design. Reseting one of the SABRE Lite boards to

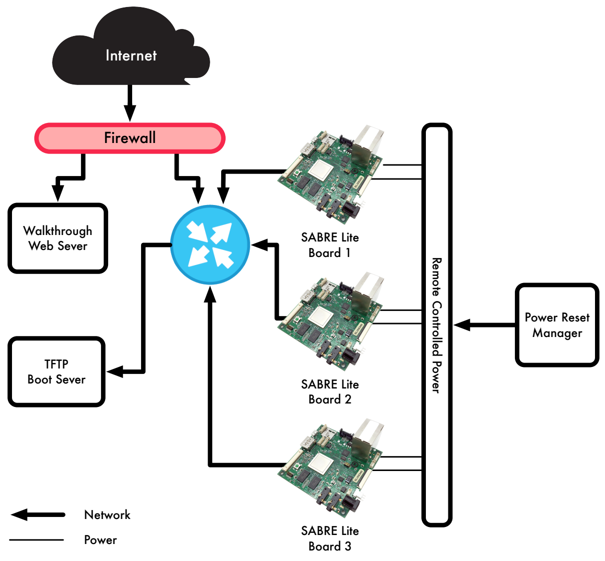


Figure 1: Physical layout of the sandbox. Remote power controls and remote boot via TFTP allow for a clean reset of the boards.

a clean state requires a reliable method for restarting the device and loading a pristine boot image. Furthermore, such reseting must be automated. The reliable restart is achieved by connecting all the boards to remote controlled power that can be programmatically managed. To ensure that the firmware booted is pristine and unmodified, each board's boot loader is programmed to boot from a network TFTP server rather than local storage, which could be modified once users obtain root access as part of the Automotive Exploitation Sandbox experience.

Now that reseting each board is automated, the question remains as to how often to reset a board. Ideally, each user of the Automotive Exploitation Sandbox would be able to reserve a particular board and have it reset once the user finishes the sandbox exercise. Unfortunately, this could lead to unintentional or even intentional denial of service if users do not relinquish their reserved devices. For the initial design, a simple timer is implemented, with a count down displayed for each board allowing users to start with a board that has sufficient time remaining to complete the sandbox exercise before its next scheduled reset. This approach does have the downside of requiring users to complete the entire exercise in one session or being forced to start fresh. The base count down time is set based on timing how long it took some beta testing nontechnical users to complete the exercise.

As a sandbox with public users, the devices and network must be isolated for security purposes. As shown in Figure 1, the design includes a firewall, which is configured limit traffic to the expected ports used in the directions. The servers hosting the instructions, TFTP server, and remote power controls are all further isolated from the actual SABRE Lite boards. This reduces the probability of a malicious user jumping from one of the intentionally vulnerable sandbox devices to a server controlling parts of the Automotive Exploitation Sandbox. As a final layer of defense to protect from the scenario where the sandbox infrastructure is compromised, the entire firewall and network are isolated and unconnected from any organization network.

While fully automated with the automatic reset of the sandbox devices, if something does go wrong manual intervention would be required. In order to alert the system administration whenever such intervention is required, an automatic monitoring system is in place. This system monitors for both device downtime and potential abuse alerting the system administrator in either case.

3 Exploitation Walkthrough

The accompanying presentation includes a live demonstration of the Automotive Exploitation Sandbox. In this section, we will go through a brief overview of the exploitation process. The full instructions with step by step details, command text, and even video tutorials will be available to users via the walkthrough web server seen in Figure 1. The core walkthrough is visualized across Figures 2, 4, and 6.

3.1 Exercised Vulnerabilities Summary

While the following subsections will go into a more detailed overview of the walkthrough that users of the Automotive Exploitation Sandbox experience, this section provides a summary of the types of vulnerabilities covered in the walkthrough.

Command Injection A user inserts escape characters along with their own command into a data field that is later used to execute a server-side command thus also executing the user's command. A simple ping command field taking a user specified IP address is used for this example in the Automotive Exploitation Sandbox.

Heap Overflow A user overflows a data structure located in the heap, which is a scratch space programs use to store temporary information in memory, to overwrite data that the program is relying on to determine its control flow. In order to demonstrate a wide variety of vulnerabilities, an Echo service that is flawed with an exploitable heap overflow is added to the device. This is an

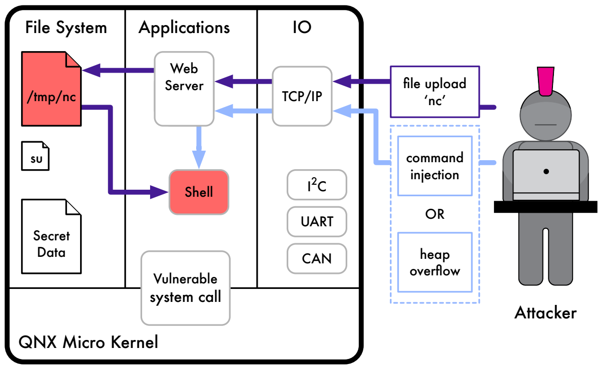


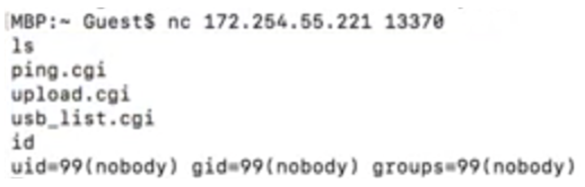
Figure 2: Overview of stage 1 of the Automotive Exploitation Sandbox. The user obtains an unprivileged remote shell.

alternate route to the first stage attack when a trivial command injection is not available. The crux of the problem is to overwrite a function pointer with the system function and then invoke it on our buffer. We designed the Echo service so as to make this attack easy. For the Automotive Exploitation Sandbox, users will be supplied with a script that will take a command as an input and generate a suitable curl request.

Kernel Memory Modification An arbitrary kernel memory modification is a vulnerability class that allows for the user to modify arbitrary kernel memory, which can be leveraged to gain root privilege and take over the entire device. For this exercise, the vulnerability is injected as a system call that the user can leverage to gain the kernel write primitive. System calls that fail to properly verify user input are the typical attack vector for this type of attack chain.

3.2 Stage 1: Obtain Remote Shell

The first stage of the Automotive Exploitation Sandbox walkthrough instructs the user on how to obtain a user privilege remote shell on the device. Figure 2 visualizes this process. First, the user uploads the netcat binary named 'nc' to through the normal file upload feature of the web server. Then, using the synthetic vulnerability introduced into the web server cgi script, the user will inject a command to start the shell listening for remote connections via netcat. At the end of this stage, the user connects to the remote shell via 'nc' and can verify that they are running as the user 'nobody.' For an example of what the remote shell looks like, see Figure 3.



```
MBP:~ Guest$ nc 172.254.55.221 13370
ls
ping.cgi
upload.cgi
usb_list.cgi
id
uid=99(nobody) gid=99(nobody) groups=99(nobody)
```

Figure 3: Example of the user connecting to the remote shell and executing commands remotely on the device.

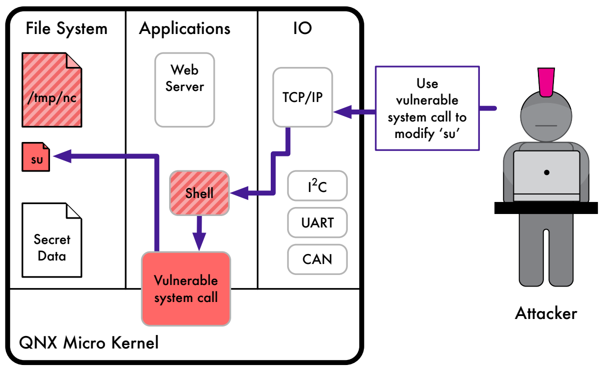


Figure 4: Stage 2 of the Automotive Exploitation Sandbox. The user exploits a synthetic vulnerable system call to modify the 'su' binary to bypass its password check.

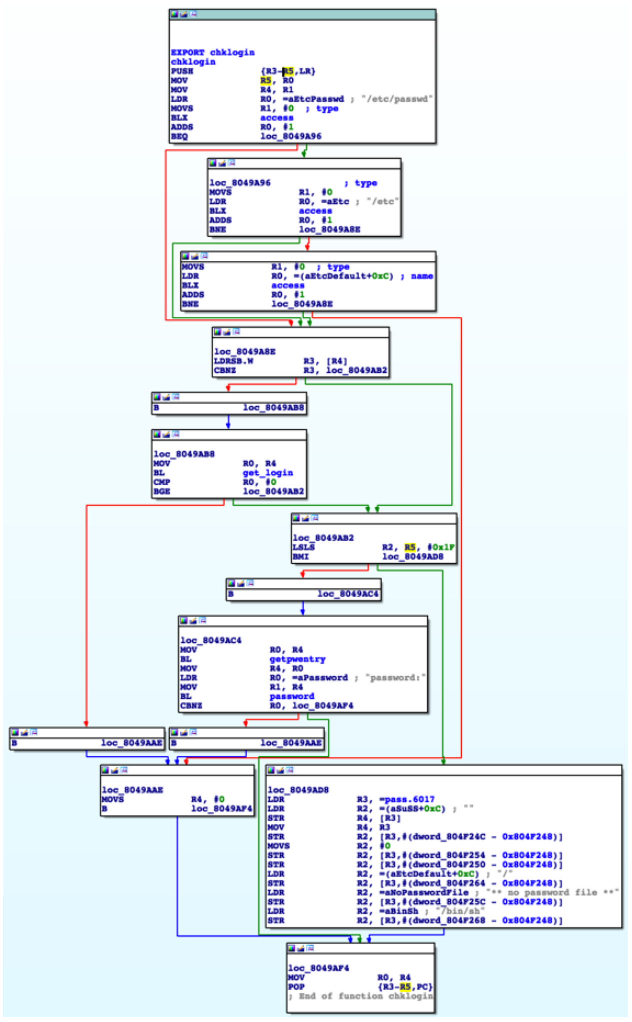
3.3 Stage 2: Privilege Escalation

The second stage leads the user through the process of escalating from a user account to a root privileged account. While the vulnerability exploited is a synthetic vulnerable system call injected into the QNX kernel, similar vulnerabilities have been reported in the past. An exploit payload program is provided to the user that can exercise this vulnerable system call to modify arbitrary memory. The details of designing and writing such a payload is left as an exercise to the user.

To escalate their user privilege, the instructions take the user through the process of modifying the 'su' binary to work without a password. The details of how an attacker would reverse engineer the 'su' binary to figure out where to patch the code is documented in depth in an optional to read document. Figure 5 is an example figure from that document showing an IDA Pro disassembly of the 'su' binary password checking function. The last step in stage 2 is for the user to use the provided payload to modify the 'su' binary via their user remote shell. The user then ends up with a root shell on the device.

3.4 Stage 3: Post Exploitation

The last stage of the Automotive Exploitation Sandbox walks the user through post-compromise attacker actions



```
EXPORT chklogin
chklogin
PUSH            {R3-R5,LR}
MOV             R5, R0
MOV             R4, R1
LDR             R0, =aEtcPasswd ; "/etc/passwd"
MOVS            R1, #0  ; type
BLX             access
ADDS            R0, #1
BEQ             loc_8049A96

loc_8049A96                 ; type
MOVS            R1, #0
LDR             R0, =aEtc ; "/etc"
BLX             access
ADDS            R0, #1
BNE             loc_8049A8E

MOVS            R1, #0  ; type
LDR             R0, =(aEtcDefault+0xC) ; name
BLX             access
ADDS            R0, #1
BNE             loc_8049A8E

loc_8049A8E
LDRSB.W         R3, [R4]
CBNZ            R3, loc_8049AB2

B               loc_8049AB8

loc_8049AB8
MOV             R0, R4
BL              get_login
CMP             R0, #0
BGE             loc_8049AB2

loc_8049AB2
LSLS            R2, R5, #0x1F
BMI             loc_8049AD8

B               loc_8049AC4

loc_8049AC4
MOV             R0, R4
BL              getpwentry
MOV             R4, R0
LDR             R0, =aPassword ; "password:"
MOV             R1, R4
BL              password
CBNZ            R0, loc_8049AF4

B               loc_8049AAE

B               loc_8049AAE

loc_8049AAE
MOVS            R4, #0
B               loc_8049AF4

loc_8049AD8
LDR             R3, =pass.6017
LDR             R2, =(aSuSS+0xC) ; ""
STR             R4, [R3]
MOV             R4, R3
STR             R2, [R3,#(dword_804F24C - 0x804F248)]
MOVS            R2, #0
STR             R2, [R3,#(dword_804F254 - 0x804F248)]
STR             R2, [R3,#(dword_804F250 - 0x804F248)]
LDR             R2, =(aEtcDefault+0xC) ; "/"
STR             R2, [R3,#(dword_804F264 - 0x804F248)]
LDR             R2, =aNoPasswordFile ; "** no password file **"
STR             R2, [R3,#(dword_804F25C - 0x804F248)]
LDR             R2, =aBinSh ; "/bin/sh"
STR             R2, [R3,#(dword_804F268 - 0x804F248)]

loc_8049AF4
MOV             R0, R4
POP             {R3-R5,PC}
; End of function chklogin
```

Figure 5: IDA Pro disassembly of the 'su' binary password verifying function.

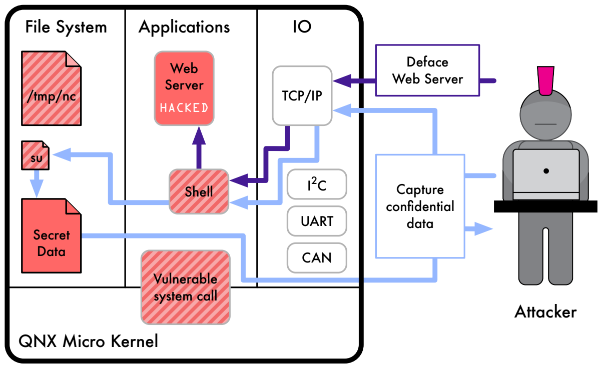


Figure 6: Stage 3 of the Automotive Exploitation Sandbox. The user is walked through defacing the web server and obtaining the secret data.

including obtaining secret data and modifying the web server content. Notably missing is installing a persistent rootkit. This typical attacker behavior is left out of the exercise as by design the devices are restored to a pristine state each reboot so that the next user can go through the exercise. A future version of the exercise with a separate on demand reseting of devices could provide such a scenario. Figure 6 illustrates these two attacker goals.

The user is walked through the process of using the same memory modifying payload to change memory in the web server changing the content served. This lets the user have a visual impact posting whatever text they desire on the device web page. After this portion of the exercise, a number of users experience an eureka moment seeing the effect of their actions and thus understanding how a malicious actor can do the same. Additionally, a secret data file is left on the device with root only read permissions so that now with root access the user can read the secret data.

4 Future Work

The Automotive Exploitation Sandbox could be built upon as a testbed to measure and verify various defensive technologies. In its current single device design, it could provide a testing platform to demonstrate host-based defenses. Expanding the sandbox into a multi-device testbed could allow for firewall and network defenses to be tested. To fully test various defenses, additional classes of attacks and payloads will be required. Feedback on the Automotive Exploitation Sandbox is welcome at sandbox-support@redballoonsecurity.com in order to further refine it and make it as useful to the community as possible.

5 Conclusion

Overall, while a fairly simple scenario, the Automotive Exploitation Sandbox is capable of walking nontechnical users through the typical steps of remote exploit, privilege escalation, and malicious actions to educate users on what such processes might look like on real hardware. The hands-on experience provided by the Automotive Exploitation Sandbox can be an eye opening experience for users without previous security experience. Even advanced users can also enjoy the sandbox experience as its real hardware and software allow exploration and deviations from the standard walkthrough.

6 Access

Visit https://sandbox.redballoonsecurity.com for access. Public access will be available after the presentation.