

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

APPLE INC.,

Petitioner,

v.

SPEIR TECHNOLOGIES LTD.,

Patent Owner.

Case IPR2023-00542

Patent 7,765,399

**DECLARATION OF ANTHONY DEROSA IN SUPPORT OF
PATENT OWNER'S RESPONSE TO THE PETITION**

TABLE OF CONTENTS

I.	BACKGROUND	1
A.	Qualifications	1
B.	List of Materials Considered.....	4
C.	Compensation.....	5
II.	LEGAL PRINCIPLES	6
A.	Claim Construction	6
B.	Burden of Proof.....	6
C.	Anticipation.....	6
D.	Obviousness	7
III.	PERSON HAVING ORDINARY SKILL IN THE ART	8
IV.	PRIORITY DATE FOR THE PATENT.....	9
V.	REVIEW AND ANALYSIS.....	9
A.	Review of the '399 Patent.....	9
B.	Summary of Opinions	10
C.	Secure Computing Primer.....	11
D.	The '399 patent discloses a secure HMI <i>exclusively</i> between a user and the secure processor.	16
E.	Marek discloses a software component called Sware that executes on the non-secure processor.....	18
F.	Marek does not disclose a secure HMI configured for enabling bi-directional communication of classified information exclusively between a user and a secure processor.	26

a. Communication is not exclusively between a user and a secure user processor	26
b. Marek's human/machine interface is not secure	28
VI. CONCLUSION	34

I, Anthony DeRosa, declare as follows:

I. BACKGROUND

1. My name is Anthony DeRosa. I have been retained by Patent Owner Speir Technologies Ltd. to provide my technical opinions regarding U.S. Patent No. 7,765,399 (the '399 patent), the Petition for *inter partes* review (IPR) numbered IPR2023-00542, and the declaration and testimony of Dr. Sandeep Chatterjee related to this IPR.
2. I submit this declaration in support of Patent Owner's Preliminary Response. I reserve the right to supplement my opinions, should I be requested to provide additional analysis in connection with the patentability of the claims of the '399 patent.

A. Qualifications

3. I am an expert in the field of computer engineering, and I specialize in the analysis and secure design of embedded computing devices and their networks. I graduated with honors from Messiah College in 2003 with a bachelor's degree in engineering. I later attended Johns Hopkins University, where I received an M.S. degree in Electrical and Computer Engineering with a concentration in signal processing in 2009.
4. I have worked in the field of computer software, firmware, and embedded systems for over twenty years, gaining experience that spans from the hardware level up through the entire software stack.

5. I have spent most of my career working in secure facilities, which are buildings, or certain areas of a building, that are specially designed to handle sensitive, classified information. In the offices in the secure facility where I worked, there were typically two phones at each desk, a “red” phone for discussing classified information and a “black” phone for discussing unclassified information. I also have experience using STE (Secure Terminal Equipment) phones, which are secure telephones used by the U.S. Government for classified, encrypted communications. A STE phone can be used to make normal, unclassified calls or classified, encrypted calls over the public phone network, depending on which mode the device is in.
6. I started working full time in 2003 as a computer engineer in the Department of Navy as a software developer in a flight simulation laboratory. I wrote software in a variety of programming languages to support the operations of the flight simulators. In my role, I worked with many types of computer architectures, programming languages and operating systems.
7. In my job for the Department of Navy, my main tasks were focused on transitioning old software, written in outdated programming languages, such as FORTRAN, which ran on ancient systems, such as VAX/VMS, to modern programming languages running on modern PC-based COTS hardware.

8. After my time as a civilian in the federal government at the Department of Navy, I continued supporting the Department of Defense as a federal contractor working for several commercial companies. During this time, I analyzed many kinds of embedded systems and devices to identify potential security weaknesses in their design and implementation. I developed proof-of-concept manipulations against these systems to show how the weakness could be exploited by an attacker.
9. Through these jobs, I gained experience in a wide range of computer architectures and instructions sets, such as Intel x86, ARM, MIPS, PowerPC, NIOS II, MicroBlaze and other custom architectures. I have designed and developed software device drivers for many types of hardware, including keyboards, UARTs, graphical displays, wired and wireless network interface controllers, compact flash cards, audio interfaces, and disk drives. I also have experience working with many types of peripheral busses, such as PCI, USB, SPI, and I²C.
10. In more recent years, I have been involved in the analysis of computer systems such as automotive applications, medical devices, and consumer electronics devices, including smart phones, TVs, tablets, printers, and network appliances. I have analyzed the software and firmware of such devices to show how these devices could be exploited by an attacker. For example, in the past

year, I showed how the security protections of a certain medical device could be circumvented by an attacker for a litigation matter, and I showed how the Digital Rights Management (DRM) protection of a video streaming platform could be circumvented (with permission of the owner) by an attacker for another litigation matter.

11. I have also analyzed the firmware and software of these kinds of systems to show how these kinds of devices infringe or do not infringe certain patents. This analysis has been performed in support of both active patent litigation as well as patent licensing negotiations.
12. Over the past few years, I have focused extensively on analyzing the implementation of secure computing systems that make use of technologies such as secure boot, ARM TrustZone, and Trusted Execution Environments (TEEs).
13. In addition to the information provided in this section, additional relevant information related to my education and professional experience is provided in my curriculum vitae, attached to my declaration as Appendix 1.

B. List of Materials Considered

14. In forming my opinions, I have relied on my education, knowledge of computer software, embedded systems, information security, and related experience. I have also considered the viewpoint of a person having ordinary

skill in the art at the time of the earliest claimed priority date for the '399 patent. I have reviewed the documents cited throughout this declaration and the documents cited in Dr. Chatterjee's declaration, which include the '399 patent (Ex. 1001), the file history of the '399 patent (Ex. 1002), Dr. Chatterjee's declaration (Ex. 1003), and each of the alleged prior art documents (Exs. 1004-1012, 1014-1027).

C. Compensation

15. I am the Managing Member and Principal Engineer at Syscall 7, LLC, an engineering services firm headquartered at 10015 Old Columbia Rd, Suite B215, Columbia, MD 21046. I am a W2 employee of Syscall 7, LLC as well as the majority owner of the company.
16. I have been retained in this matter through RMB Consulting, which has contracted my services through Syscall 7, LLC. RMB Consulting is compensating Syscall 7, LLC for my time and expertise provided in connection with this engagement at my normal rate. I have received no additional compensation from Syscall 7, LLC or RMB Consulting for my work in this matter other than the hourly rate used for all work at Syscall 7, LLC, and my compensation does not depend on the contents of this report, any testimony I may provide, or the ultimate outcome of this litigation.

II. LEGAL PRINCIPLES

A. Claim Construction

17. I understand that the first step in performing a validity analysis of the patent claims is to interpret the meaning and scope of the claims by construing the terms and phrases found in those claims. I understand that the appropriate construction of a claim term is its ordinary and accustomed meaning as understood by one of ordinary skill in the art at the time of the invention in the context of the entire patent.
18. I understand that standard for claim construction in an *inter partes* review is the same standard as is applied in district court proceedings.
19. I understand that a determination of the meaning and scope of the claims is a matter of law. I have been informed that to determine the meaning of the claims, one should consider the intrinsic evidence, which includes the patent's claims, written description, and prosecution history.

B. Burden of Proof

20. I understand that in an *inter partes* review, the petitioner has the burden of proving unpatentability by a preponderance of the evidence.

C. Anticipation

21. I have been instructed by counsel and understand that a reference is anticipated if a single prior art reference discloses each and every claim element, either explicitly or inherently, as arranged in the same way as in the

claim. I understand that where even one claim element is not disclosed in a reference, a contention of anticipation fails.

22. I further understand that when a reference fails to explicitly disclose a claim element, that reference inherently discloses that element only if the reference must *necessarily* include the undisclosed claim element.

D. Obviousness

23. I have been instructed by counsel and understand that a combination of prior art references may render a claim obvious if, at the time of the invention, a person of ordinary skill in the art would have selected and combined those prior-art elements in the normal course of research and development to yield the claimed invention.
24. I understand that in an obviousness analysis, one should consider the *Graham* factors: the scope and content of the prior art; the differences between the claimed inventions and the prior art; the level of ordinary skill in the art; and certain secondary considerations, identified below. I further understand the obviousness analysis is to be performed on a claim-by-claim basis. I understand that a person of ordinary skill in the art is a person of ordinary creativity, not an automaton.
25. I have been instructed by counsel and understand that obviousness requires more than a mere showing that the prior art includes separate references

covering each separate limitation in a claim under examination. I understand obviousness requires the additional showing that a person of ordinary skill at the time of the invention would have been *motivated* to combine those references in a manner that would include all limitations of the challenged claim, and, in making that combination, a person of ordinary skill in the art would have had a *reasonable expectation of success*.

26. I also understand that an obviousness analysis must be conducted with awareness of the distortion caused by hindsight bias and with caution of arguments reliant upon *ex post* reasoning. For instance, I understand that when considering obviousness, I should put myself in the position of a person of ordinary skill in the field at the time of the invention, rather than considering new information that is known today but was not known before the priority date of the challenged patent.
27. In particular, I understand that it is improper to use the challenged patent's disclosure or invention as a roadmap to find its prior-art components, because such an approach discounts the value of combining various existing features or principles in a new way to achieve a new result.

III. PERSON HAVING ORDINARY SKILL IN THE ART

28. The field of technology for the '399 patent is related to secure mobile computing systems. In my opinion, a person of ordinary skill in the art of the

subject matter of the '399 patent would have been a person having the equivalent of an undergraduate degree in electrical and computer engineering (or a related field) and approximately two years of work experience in the field of embedded systems. Additional education in the field could be substituted for industry experience, and vice versa. Under this standard, I am one of at least ordinary skill in the art.

29. As of the filing date of the '399 patent, I was at least a person of ordinary skill in the art relating to the '399 patent.

IV. PRIORITY DATE FOR THE PATENT

30. I understand that the priority date for the '399 patent is in the early 2000s timeframe, specifically, February 22, 2006. I have used this date in determining the level of skill and knowledge of the relevant person of ordinary skill, which I discussed above.

V. REVIEW AND ANALYSIS

A. Review of the '399 Patent

31. I have thoroughly reviewed the '399 patent, including the specifications, claims, and prosecution history. The '399 patent discloses a mobile computing device with an architecture that allows for the secure storage, processing, and communication of both classified and unclassified data. Ex. 1001 at Column 1, Lines 7-10. The device incorporates a two-processor system comprising a secure processor and a non-secure processor. Ex. 1001 at Column 2, Lines 23-

40 The secure processor is designed to handle sensitive classified information and operates with a trusted operating system and applications. It interfaces with a cryptographic engine that encrypts sensitive data before it can be transferred to the non-secure processor or communicated externally. Id.

32. In contrast to the secure processor, the non-secure processor is intended for unclassified data and utilizes commercial off-the-shelf (COTS) hardware and software to provide a familiar user experience for non-sensitive operations. Id.

33. The mobile computing device also features a secure Human-Machine Interface (HMI) for communicating bi-directional communication exclusively between a user and the secure processor, which provides a trusted pathway for user inputs and system outputs, protecting against unauthorized monitoring from the non-secure environment. Ex. 1001 at Column 3, Lines 4-10, Figs. 3-5.

B. Summary of Opinions

34. I understand that Petitioner alleges that various combinations of references render obvious claims 1-7 and 9-10 (the Challenged Claims) of the '399 patent. Pet. at 2. I disagree. In particular, I disagree that the Marek-Skarine, Marek-Skarine-Yoon, or Marek-Skarine-Moon combinations render any of the Challenged Claims of the '399 patent obvious.

35. I understand that Dr. Chatterjee opines that claims 1-7, 9-10, 12, 13, and 15-18 are obvious over Marek and Skarine; that claims 4-5 are obvious over Marek, Skarine, and Moon; and that claims 9-10 are obvious over Marek, Skarine, and Moon. Ex. 1003 at ¶16. I disagree with these opinions as to claims 1-7 and 9-10, and I have not been asked to opine as to claims 12, 13, and 15-18, which I understand were not challenged in the IPR petition.

C. Secure Computing Primer

36. Information security and secure computing technologies have seen significant advancements over the last few decades. In early days, processors executed very basic operating systems with minimal security features. For example, there was no memory protection between individual processes, which meant that a bug or malicious process could easily interfere with or corrupt the memory used by other processes running on the same processor, leading to system crashes or serious vulnerabilities.
37. As secure computing evolved, the concept of memory segmentation and paging was introduced, which divides memory into segments with different permissions, preventing processes running on the same processor from accessing each other's memory space. Paging further refined this by allowing memory to be divided into pages, which each page having its own permissions. For example, some pages can be marked with the NX ("No

Execution”) permission, which prevents the processor from executing that region of memory.

38. The concept of “privilege rings” was also introduced by modern processors to establish different levels of privilege for processes running on the system. Most processor architectures today implement this concept, with “Ring 0” having the highest privilege and subsequent rings (1, 2, 3, etc.) having progressively fewer privileges. The privilege levels are enforced by the processor at the hardware level, and there are strict hardware-enforced rules governing how software switches between these levels.
39. The operating system (OS), or kernel, executes in Ring 0. In this mode, the OS has unrestricted access to the system and can directly interact with hardware and system resources. In contrast, user space applications execute in Ring 3 and have limited privileges. Processes in this ring cannot directly access hardware or reference memory outside of their allocated space. This separation helps prevent applications from accidentally or maliciously modifying kernel memory and compromising the system.
40. Despite these and other sophisticated security measures, systems may still be vulnerable to being compromised by malware. Malware, short for malicious software, is designed to damage, disrupt, or gain unauthorized access to a

computer system. Malware authors use a variety of techniques to circumvent security measures, some of which include:

- a. **Exploiting Vulnerabilities:** Malware often seeks out vulnerabilities in software or firmware that can be exploited to bypass security measures. For example, a buffer overflow vulnerability could be used to bypass the NX (“No Execute”) bit protection by executing malicious code in a memory space that should be non-executable.
- b. **Memory Corruption Attacks:** These attacks aim to corrupt the normal behavior of applications or services in a system, leading to arbitrary code execution. Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP) are security measures designed to prevent such attacks, but vulnerabilities or implementation flaws can still be exploited by sophisticated malware.
- c. **Social Engineering and Phishing:** These techniques trick users into compromising security themselves, for example by convincing them to disable security software, click on malicious links, or divulge sensitive information.
- d. **Privilege Escalation:** Malware may initially execute in a low-privilege context (like Ring 3 in the privilege rings system). It will

then attempt to exploit a vulnerability to gain higher privileges (such as Ring 0), allowing it to have unrestricted access to the system.

41. The ultimate exploitation of a computer system provides “root” level access, which equates to Ring 0, or kernel mode as it is also called. On Unix-like systems, the “root” user is the administrator with complete control over the system, hence the term "root access."
42. Malware that has gained root access has essentially hit the jackpot in terms of what it can do to a system. Root access grants the malware full control over the system, allowing it to perform a wide range of malicious activities without any restrictions. Below are some of the malicious operations that malware can perform with root access:
 - a. **Spying and Surveillance:** Malware can monitor user activity, capture keystrokes, take screenshots, activate microphones or cameras for eavesdropping, and steal sensitive information.
 - b. **Manipulate Processes and Services:** Malware can start, stop, or modify system processes and services, including those related to security and system maintenance.
 - c. **Manipulate Data:** Malware with root access can read, modify, delete, or encrypt files and data. This can lead to data breaches, data

corruption, or ransomware attacks where attackers demand payment for the decryption key.

- d. **Install Additional Malware:** It can download and install additional malware onto the system, including Trojans, keyloggers, or more sophisticated rootkits.
- e. **Create Backdoors:** Malware can create hidden backdoors that allow attackers to regain access to the system even after the initial malware has been removed.
- f. **Control Over Hardware:** With root access, malware can interact directly with hardware, potentially leading to physical damage, overheating, or exploitation of hardware vulnerabilities.
- g. **Modify System Configuration:** It can alter system settings, corrupt or disable system utilities, and change network configurations, potentially leading to a system crash or making the system part of a botnet for coordinated attacks.
- h. **Disable Security Software:** It can disable antivirus programs, firewalls, and other security measures, making the system more vulnerable and preventing the detection of malicious activities.

- i. **Covering Tracks:** It can modify or delete logs, hide its presence, and use sophisticated techniques to remain undetected, making forensic analysis and system recovery challenging.

D. The '399 patent discloses a secure HMI *exclusively* between a user and the secure processor.

- 43. As discussed in the primer section on secure computing, malicious software has the capacity to infect a system and gain execution at all processor privilege levels. This “root” access affords the attacker all kinds of opportunities to subvert the system, including monitoring or manipulating all activity on the processor.
- 44. The '399 specification underscores the potential for malware to infect a system and describes how commercial-off-the-shelf (COTS) software running on the non-secure processor is susceptible to these kinds of attacks.

For example, COTS operating systems and applications may not be sufficiently trustworthy for handling this type of information. **Such programs can be susceptible to being compromised by various means including hacker attacks, viruses, worms, Trojan horses, and a wide variety of other means that are known to those skilled in the art.**

Ex. 1001 at Column 1, Lines 50-55

45. For this reason, the '399 specification teaches that classified information should never be exposed to the non-secure processor except in encrypted form.

Further classified data from the secure processor via the cryptographic engine **can be communicated to the non-secure processor exclusively in an encrypted form.**

Ex. 1001 at Column 2, Lines 65-67

46. A non-secure processor cannot be trusted because it may have been compromised by an attacker and may be running malicious software at any privilege level. Furthermore, the malicious software may be sophisticated enough to hide its tracks and eliminate any visible evidence of its presence. As the '399 patent recognizes (e.g., by enforcing encryption and including an exclusive connection between the user and the secure processor) one way to improve the safeguards for classified data is to ensure that the non-secure processor never has access to the unencrypted, or “plaintext”, form of the classified data.
47. The '399 specification discloses an apparatus and method for protecting classified data even in the scenario where the non-secure processor is fully compromised by an attacker. This is accomplished by means of a secure HMI

that enables bi-directional communication *exclusively* between a user and the secure processor.

48. The '399 specification describes that secure HMI devices “typically can include one or more features to ensure trusted communications between the user and the secure processor.” Ex. 1001 at Column 7, Lines 40-44. It further explains that “secure HMI 308 can provide a trusted path to applications executing on the secure processor 302” which can “prevent invasive or unauthorized applications from monitoring user inputs and system outputs.” Ex. 1001 at Column 7, Lines 44-47.

E. Marek discloses a software component called Scware that executes on the non-secure processor.

49. Marek describes Scware as “software that acts as a trusted agent for the SCM 702 on the untrusted host environment.” 10:42-45. Marek further states that “Scware 724 resides on the untrusted host environment while the SCM 702 is operating” and that “Scware 724 implemented in conjunction with TSM 704 creates secure I/O paths 730 of input and output devices 722, 720 of the untrusted host 718.” Ex. 1004 at Column 10, Lines 47-51.
50. Marek teaches that Scware acts a device driver for the input and output devices, such as a keyboard and display.

In H-mode, only I/O interfaces allowed by the Scware 724 may be implemented and no other applications may be allowed to run on the host platform. The Scware 724 **manages input from an input device (e.g. keyboard) and output to an output device (e.g. display device)** while the host OS is completely bypassed. As a result, the Scware 724 **operates as a dedicated input/output device driver** (e.g. keyboard/display driver).

Ex. 1004 at Column 10, Lines 58-65.

51. As a POSITA would have understood, a device driver is a specialized type of computer software that acts as an intermediary or translator between the system-level software (such as the operating system) and a hardware device (such as a keyboard). The device driver enables communication between the system and the device, ensuring that the system can make effective use of the hardware device.
52. Device drivers typically interact with hardware devices through a combination of low-level mechanisms.
 - a. **I/O Ports:** Many devices are mapped to specific Input/Output (I/O) ports. The device driver sends and receives data to and from the hardware by reading from and writing to these I/O ports using special CPU instructions like IN and OUT in the Intel x86 architecture.

- b. **Memory-Mapped I/O:** Instead of or in addition to I/O ports, some devices use regions of memory addresses that are mapped for data transfer. The device driver can read and write to these memory addresses to control the device and transfer data. Instead of reading or writing to system DRAM, the data is read or written directly from or to the registers of the hardware device.
- c. **Interrupts:** Hardware devices can signal the CPU by triggering interrupts. These interrupts inform the device driver that the device needs attention, such as when new data has arrived or a device operation has completed. The device driver includes interrupt service routines (ISRs) that handle these interrupts and perform the necessary actions.
- d. **Direct Memory Access (DMA):** For high-speed data transfers, such as those needed for disk operations or network communications, devices often use DMA. DMA allows a device to transfer data directly to and from memory without involving the CPU for each byte. This means the CPU does not need to stop to perform the copy and speeds up the performance of the whole system. The device driver sets up and manages these DMA transfers.

53. Device drivers are sophisticated software components that must know the low-level details of how to interact with the hardware and abstract the capabilities of the device in a way that the system software operating above the device driver does not need to know about all the low-level details.
54. Marek teaches that Sware 724 acts as a device driver that, in conjunction with the TSM 704, creates I/O paths 730 of input and output devices 722 (keyboard) and 720 (display) of the untrusted host 718. Ex. 1004 at Column 10, Lines 49-51. A POSITA would have understood that Sware's role in creating an IO path is typical for a device driver. For example, when a key on keyboard 722 is pressed, a POSITA would understand that the keyboard controller hardware would detect the key press and send an interrupt to the non-secure processor of the untrusted host, on which Sware is executing. A POSITA would also understand that, in response to this hardware interrupt, an Sware interrupt subroutine would read from the standard keyboard controller IO ports to identify which key on the keyboard was pressed and then read data from the given IO port into one of the untrusted processor's registers. In other words, user information entered by keyboard 722 would be temporarily saved in a register on the untrusted host. Although device driver implementation details may vary, a POSITA would recognize that

information or data received from the keyboard is stored on the untrusted host in some capacity.

55. Marek also describes that the Sware would also act as a device driver for a display 720. Based on Marek's description, including Figure 7 of Marek, a POSITA would also have understood that user information being sent to display 720 from the SCM would first be written to frame buffer memory within the untrusted host, the contents of which is used to drive the display 720. In other words, to put characters or graphics on the display, the Sware must write the desired data to the frame buffer memory.

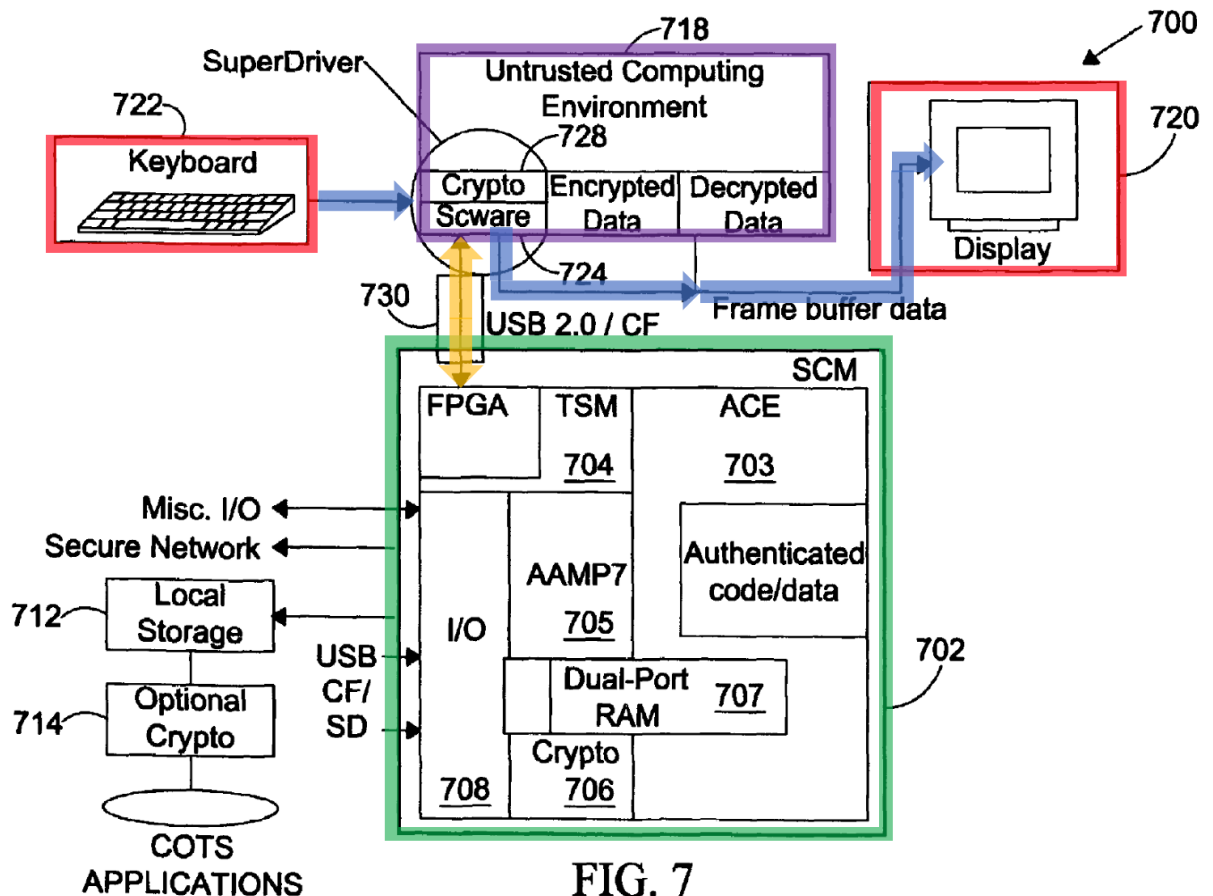


FIG. 7

56. Marek’s description that “[w]hen H-mode is finished, the Scware 724 zeroizes the appropriate memory regions of the host platform, including the output device’s frame buffer” (Ex. 1004, 11:7-12) provides further evidence that user information bound for display 720 is sent first to memory regions within the untrusted computing environment. A POSITA would readily recognize that zeroizing those memory regions after the data transfer is finished would be done in an effort to remove confidential user data that had been stored in those memory regions.
57. In both examples—i.e., user information input from keyboard 722 or user information output to display 720—the non-secure processor is actively involved in processing the data between the user and the trusted computing environment.



Illustration of Information Flow in Marek

58. Marek’s description of the Scware driver functionality is consistent with typical keyboard driver software functionality at the time the ’399 patent was filed. At that time, PS/2 keyboards were still in widespread use, especially in

US Department of Defense government offices. A diagram showing how a PS/2 keyboard interfaces with a traditional PC system is shown below in Figure 1.

59. The keyboard (see the purple box in the diagram) is plugged into the motherboard through a 6-pin mini-DIN connector. The signals in the connector are routed to an Intel 8042 keyboard controller (KBC) on the motherboard of the PC (see yellow box).

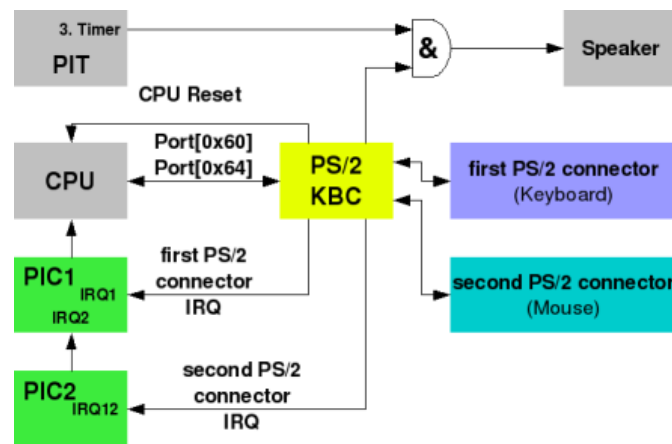


Figure 1 Overview of the PS/2 Keyboard Controller¹

60. When a key on the keyboard is pressed, the keyboard controller detects the key press and sends an interrupt to the CPU via the chipset's Programmable Interrupt Controller (PIC), which is shown in the green box. The CPU, which is the non-secure processor in Marek, is notified of this interrupt and subsequently reads IO ports 0x60 and 0x64 to retrieve the keyboard scan codes that indicate what key was pressed.

¹ https://wiki.osdev.org/'8042'_PS/2_Controller

61. The device driver software (e.g., Scware in Marek), running on the CPU, would execute an interrupt subroutine (ISR) in response to the keyboard's interrupt. The ISR would read from the standard keyboard controller IO ports, which are 0x60 for data and 0x64 for command/status, to identify which key on the keyboard was pressed.
62. Note that the IO ports can only be read using the "IN" instruction of the x86 instruction set. The "IN" instruction reads data from the given IO port into one of the processors registers.
63. Marek's description of the Scware driver functionality is also consistent with typical display driver software functionality at the time the '399 patent was filed. For example, a VGA controller, which is shown in blue in the figure below, reads the display data out of the framebuffer and sends the appropriate signals to the external display device, such as a VGA monitor.

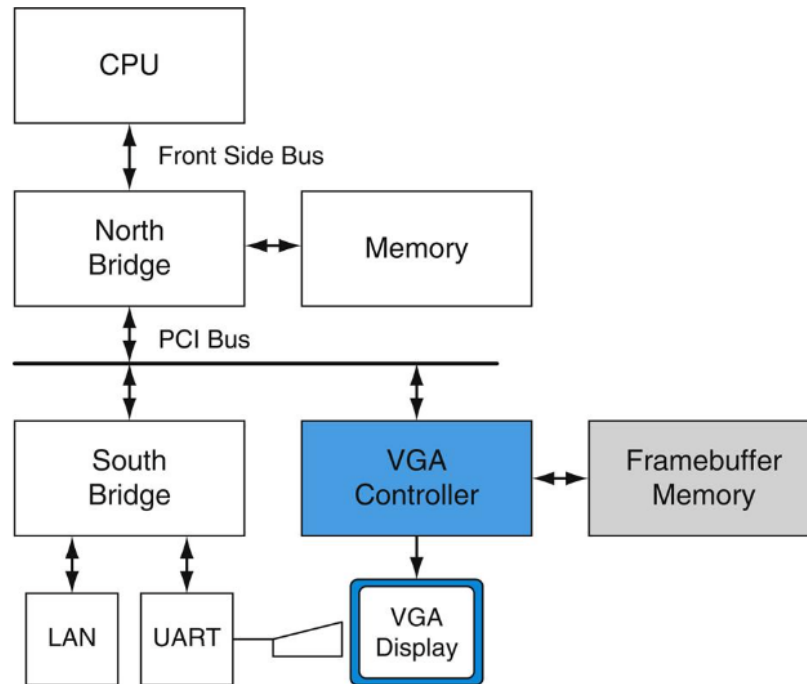


Figure 2 Block Diagram of Legacy PC²

F. Marek does not disclose a secure HMI configured for enabling bi-directional communication of classified information exclusively between a user and a secure processor.

a. Communication is not exclusively between a user and a secure user processor

64. In support of his opinion that the Marek-Skarine combination discloses or renders obvious a “secure human/machine interface configured for enabling bi-directional communication of classified information exclusively between a user and said secure user processor,” Dr. Chatterjee identifies Marek’s keyboard 722 and display 720 as a “secure human/machine interface.” Ex.

² Computer Organization and Design RISC-V Edition, Patterson and Hennessy, 2017, Appendix B

1003 at ¶108. Dr. Chatterjee also identifies the Scware software that resides on the untrusted host in Marek’s computing environment. Dr. Chatterjee says that “when the SCM 702 is in operation, information input into the input device 722 (e.g., by a user) bypasses the untrusted host 718 and instead is provided to the SCM 702 along a secure I/O path 730; similarly, information from the SCM 702 is provided to the output device 720 (and thus to a user) via a secure I/O path 730, again bypassing the untrusted host 718.” Ex. 1003 at ¶109 (emphasis added). I do not agree. As discussed above, Marek’s description of the Scware device driver software residing on the untrusted host makes quite clear that user information input into keyboard 722 does not bypass the untrusted host 718. Indeed, all user information from keyboard 718 is processed by hardware within the untrusted host 718 prior to being sent to the SCM. Similarly, all user information being sent from the SCM to display 720 is processed by hardware within the untrusted host 718, and does not bypass the untrusted host 718.

65. Dr. Chatterjee also states that “Marek’s input and output devices 722, 720 can be considered to be a secure human/machine interface at least because they are an interface by which a user inputs information to the SCM 702 (via the input device 722) and receives information from the SCM 702 (via the output device 720).” Ex. 1003 at ¶110. I do not agree. Serving as an interface by

which a user inputs information to the SCM and receives information from the SCM does not cause Marek's input and output devices to be a secure human/machine interface.

66. I also disagree with Dr. Chatterjee's assertions that the communication enabled by input and output devices 722 and 720 in Marek is exclusively between the user and the secure user processor because, according to Dr. Chatterjee, "the functionality of the untrusted host device is disabled." Ex. 1003 at ¶114. Data from the keyboard and to the display in Marek is communicated to the untrusted host while Scware is running. Ex. 1004 at Column 10, Line 31-Column 11, Line 15. This does not change when the host OS or applications running on the host OS are disabled. I also disagree that enabling communication exclusively between the user and the secure user processor would have been obvious in light of Marek's disclosure, as Dr. Chatterjee asserts in paragraph 114. I note that Dr. Chatterjee provides no basis for his opinion that this limitation would have been obvious, and I reserve the right to respond if he later offers such an opinion.

b. Marek's human/machine interface is not secure

67. Dr. Chatterjee opines that "Marek's input and output devices 722, 720 can be considered to be a secure human/machine interface at least because they are an interface by which a user inputs information to the SCM 702 (via the input

device 722) and receives information from the SCM 702 (via the output device 720).” Ex. 1003 at ¶ 110. He also opines that this interface “is secure (or at least a POSITA would have found this to be obvious) at least because it is an interface to the ‘Secure COTS computing environment’ provided by the SCM 702 of the combination.” Ex. 1003 at ¶ 161. I disagree, as explained below.

68. Marek’s input and output devices 722 and 720 (keyboard and display in Figure 7) are not secure. Rather they are vulnerable to attack and compromise of the non-secure processor. If an attacker gains privileged access to the non-secure processor in Marek, she can monitor and manipulate the classified information and do so without detection. This is because in Marek, there is no disclosure of any data to/from keyboard 722 or display 720 being protected by encryption while being processed by the non-secure processor. *See* Ex. 1004 at Column 10, Line 40-Column 11, Line 14, Figure 7.
69. The functionality associated with “H-mode,” where Marek states that the host OS is completely bypassed does not transform input and output devices 722 and 720 into a secure HMI and does not create a means of communicating exclusively between the user and the SCM:

In H-mode, only I/O interfaces allowed by the Scware 724 may be implemented and no other applications may be allowed to run on the host platform. The Scware 724 manages input from an input device

(e.g. keyboard) and output to an output device (e.g. display device) **while the host OS is completely bypassed.** As a result, the Scware 724 operates as a dedicated input/output device driver (e.g. keyboard/display driver). It is to be noted that a security critical section concept is used to implement the Scware 724 in H-mode to disable all applications but the Scware 724 on the host environment.

Ex. 1004 at Column 10, Lines 58-67. The “H-mode” described in Marek is similar to an execution mode developed by Intel called System Management Mode (SMM), which was introduced by Intel in its “SL” version of the 386 processor and later became standard in their 486 and Pentium processors in 1993.

70. System Management Mode (SMM) is a special-purpose operating mode provided for handling critical, platform-specific functions, such as BIOS-level hardware control, power management and other system-level functionality. In SMM mode, the processor executes the specialized SMM software at a higher privilege level than the operating system. This privilege is sometimes referred to as Ring -2, to indicate that it is more privileged than even Ring 0 in which the operating system kernel executes. While the processor executes the specialized software in SMM mode, the operating system is in a suspended state.

71. The “H-mode” described in Marek is also similar to how a “hypervisor” works. A hypervisor is a lightweight operating-system-like component that runs below the operating system. The job of the hypervisor is to abstract away the hardware resource of the computer such that it can potentially execute more than one operating system on the same machine. A hypervisor is sometimes said to be running in Ring -1, which indicates it has higher privilege than the operating system kernel running in Ring 0, but lower privileges than SMM mode, which is sometimes referred to as Ring -2.
72. Both SMM mode and hypervisors were widely used by COTS systems at the time of Marek. Either technique could be used to implement “H-mode” for the Scware component that Marek discloses.
73. However, regardless of the specific technique used to implement the execution mode for Scware, the Scware component is still vulnerable to being compromised by malware because it executes on the non-secure processor and in the untrusted computing environment. If malware is present on the untrusted computing environment, which is more likely because the untrusted host environment is running untrusted applications and an untrusted operating system, the user information passing unencrypted through the untrusted host could be intercepted, even if Marek’s H-mode seeks to bypass or disable the host OS.

74. SMM and hypervisors are two examples of how Marek’s “H-Mode” might be implemented. While both of these techniques provide a more secure execution environment than a traditional host operating system, the weakness of these approaches, as well as the fundamental weakness of Sware in Marek, is that these environments execute on the non-secure processor, which is vulnerable to attack, even if Sware attempts to ensure that the host OS is bypassed. For these reasons, I disagree with Dr. Chatterjee’s assertion that the HMI disclosed in Marek is secure, or that a POSITA would have found this to be obvious. Ex. 1003 at ¶115. As explained above, Marek’s implementation of Sware is similar to commercially available implementations at the time and that were in fact widely used for many years after. *See, e.g.*, discussion below of vulnerabilities discovered in 2015 in open-source hypervisors.
75. As an example of potential security flaws in implementations like Sware, in 2008 Embleton and Sparks³ showed how an attacker could insert a “keylogger” into the SMM execution environment, which would provide the attacker with the ability to monitor all the keystrokes coming from the keyboard. Such an attack would circumvent the protections afforded by the

3

https://infocon.org/cons/Black%20Hat/Black%20Hat%20USA/Black%20Hat%20USA%202008/conference%20cd/speakers/Embleton_Sparks_SMM_Rookits/BH_US_08_Embleton_Sparks_SMM_Rootkits_WhitePaper.pdf

trusted computing environment of Marek because the attacker has access to the classified data before it even gets to the trusted computing environment.

76. Hypervisors are also vulnerable to malicious software, and there is a specific term for this kind of attack called “hyperjacking”. For example, a vulnerability in open-source hypervisors, such as QEMU, Xen, KVM, VirtualBox and others, was discovered in 2015 by Jason Geffner. This attack allows malicious software to “escape” from the hypervisor-constrained environment and execute unrestricted code directly on the host processor.⁴
77. Through attacks such as these against SMM and hypervisors, malicious software has the potential to exploit and corrupt the operation of anything executing in the non-secure processing environment, including Sware.
78. In contrast, the ’399 patent recites a secure HMI in which communication is exclusively between a user and a secure processor. This exclusive communication means that even if the non-secure processor is compromised by an attacker, classified information can still be securely transmitted to and from the HMI devices because the sensitive data is never exposed to the non-secure processor in its unencrypted form.

⁴ <https://www.crowdstrike.com/blog/venom-vulnerability-details/>

VI. CONCLUSION

79. I declare under penalty of perjury under the laws of the United States of America that the foregoing statements are true and correct.

Executed on November 14, 2023 at 21:20

A handwritten signature in black ink, reading "Anthony V. DeRosa", is written over a horizontal line.

Anthony DeRosa

Appendix 1

Anthony V. DeRosa
16332 Carrs Mill Rd Woodbine, MD 21797 (410) 671-5389
anthony.derosa@syscall7.com

SUMMARY

Anthony DeRosa is an information security expert specializing in the reverse engineering and vulnerability research of networking and embedded computing devices. He has over 20 years of experience in reverse engineering firmware and application software using static reverse engineering tools such as IDA Pro, Binary Ninja, and Ghidra, and he is experienced with the runtime debugging environments of GDB, WinDbg, and JTAG-based debuggers like Lauterbach, Green Hills, American Arium, and BDI. Through fuzzing and a combination of static and dynamic reverse engineering, he has uncovered serious vulnerabilities and recommended techniques for securing these systems against the weaknesses uncovered.

Mr. DeRosa is also a full stack software developer, with experience in system-level software development in C and assembly language (x86, ARM, MIPS, PowerPC, NIOS II, and MicroBlaze). With engineering experience that spans from the hardware level up through the entire software stack, he has designed and developed VHDL, debugged hardware with scopes and logic analyzers, developed firmware in C and assembly for multiple embedded platforms, developed OS-level device drivers, developed custom networking protocols, developed Windows GUIs in high-level languages, and developed cloud-based applications using the full stack of web technologies (Django, Bootstrap, AngularJS, Vue.js, MySQL)

TECHNICAL SKILLS

- Embedded and system-level software development using C and assembly (x86, ARM, MIPS, PowerPC, NIOS II, and MicroBlaze)
- Software development environments (PyCharm, gdb, Eclipse, Visual Studio, American Arium, Lauterbach, Green Hills, Xilinx, and BDI)
- Device driver development (wired and wireless network interface controllers, compact flash cards, PCI, SPI, and I²C)
- Configuring and developing custom and commercial networking protocols, including the development of Wireshark dissectors in C and Lua
- Static and dynamic reverse engineering (IDA Pro, Ghidra, Binary Ninja, WinDbg, OllyDbg)
- Scripting languages (Python, Perl, Bash, Ruby)
- Higher level languages, including Python, C++, C#, and Java development in Windows and Linux environments.

CERTIFICATIONS

- Cisco Certified Network Associate (CCNA)
- GIAC Certified Penetration Tester (GPEN)
- GIAC Certified Incident Handler (GCIH)
- GIAC Certified Firewall Analyst (GCFW)
- GIAC Certified Forensic Analyst (GCFA)
- GIAC Security Expert (GSE)
- GIAC Certified Intrusion Analyst (GCIA)
- GIAC Security Essentials (GSEC)
- Certified Ethical Hacker (CEH)
- Certified Scrum Master (CSM)

EDUCATION

M.S. IN ELECTRICAL AND COMPUTER ENGINEERING
The Johns Hopkins University GPA 4.0/4.0
May 2009
Baltimore, MD

B.S. IN COMPUTER ENGINEERING WITH C.S. DEPT. HONORS
Messiah College GPA 3.94/4.0
May 2003
Grantham, PA

EXPERT WITNESS

- Repairify, Inc., v. Keystone Automotive Industries, Inc. US District Court for the Western District of Texas. Serving as expert witness for automotive-related embedded systems analysis. Ongoing.
- Sonrai Memory Ltd. v. Apple Inc. U.S. District Court for the Western District of Texas. Performed source code review of proprietary Apple source code and documentation for the plaintiff. Ongoing.
- Speir Technologies Ltd. v. Apple Inc. U.S. District Court for the Western District of Texas. Assisted in developing infringement theory and claim charts for the plaintiff. Performed source code review of Apple source code. Ongoing.
- Tech Pats. Confidential. Consulting support for software reverse engineering and development of claim charts for patent infringement disputes. Ongoing.
- IPCom v. AT&T Corp. U.S. District Court for the Eastern District of Texas. Performed source code review for the plaintiff. Ongoing.

- Gardner v. Lyft et. al. Superior Court of Massachusetts. Consulting on mobile device authentication methods for the plaintiff. Submitted a preliminary, informal report. 2022.
- Evolved Wireless. U.S. International Trade Commission Investigation No. 337-TA-1227. Source code review of LTE baseband software. Ongoing.
- Geomatrix v. Eljen. U.S. District Court for the District of Connecticut. Performed image metadata analysis for the defendant. 2022
- Shenzhen Carku Technology Co. International Trade Commission Investigation No. 337-TA-1256. Technology consulting. 2021
- Q3 Networking. U.S. International Trade Commission Investigation No. 337-TA-1227. Consulting on wireless and wired routers and access points. 2020 – 2021.
- Cellular Communications Equipment LLC v. HMD Global Oy. U.S. District Court for the Eastern District of Texas Marshall Division. Served as testifying expert for the plaintiff, submitted a declaration, and testified at deposition in a patent infringement dispute. 2020
- Alifax Holding Spa v. Alcor Scientific Inc. et. al. U.S. District Court for the District of Rhode Island. Performed reverse engineering of a blood analysis machine as a consulting expert for the defense. 2019
- Derive Power et. al. v. EZ Lynk SEZC et. al. U.S. District Court for the District of Utah. Consulting expert for the defense in a copyright dispute. 2018
- Class v. Apple. U.S. District Court MDL 2827. Consulting support to a testifying expert for the class. 2018

EMPLOYMENT

SYSCALL 7, LLC

WOODBINE, MD

CEO and Principal Security Researcher

JANUARY

2014 - PRESENT

- Performed software analysis and reverse engineering on mobile phones, tablets, laptops, servers, and other consumer and industrial electronics devices to show evidence of use for patent litigation.
- Performed reverse engineering in support of technology-related litigation to identify the measures taken to protect trade secrets and intellectual property for embedded devices in the automotive, medical, and telecommunications industries

- Performed security research on Android-based mobile devices
- Developed the curriculum for a 3-day, hands-on course on reverse engineering and security analysis. Taught this public course to students coming from around the world for the past two years.
- Conducted vulnerability research and software security audits on optical and copper networking devices to identify security vulnerabilities in the firmware of the systems under investigation
- Developed fuzzers against networking platforms to identify potential security weaknesses in the protocol implementations on these devices
- Extracted firmware images from x86, PowerPC, and MIPS embedded devices using flash/EEPROM programmers such as DediProg and BP Microsystems in order to analyze the security of the system through a combination of static and dynamic reverse engineering
- Developed proof-of-concept manipulations to demonstrate the weaknesses uncovered on the devices under investigation
- Developed a custom SPI-based driver in C to enable port mirroring on an undocumented Broadcom chipset in order to enable further introspection of backplane traffic on an optical networking device under investigation
- Performed binary reverse engineering against custom computer systems running a variety of operating systems (Windows, Linux, VxWorks, and other proprietary systems) to identify vulnerabilities and security weaknesses
- Developed custom software tools in C, Python, and assembly to support proprietary operations
- Setup a lab environment in which to configure network infrastructure devices and simulate a variety of real-world networking scenarios. The devices in the lab included enterprise and ISP-grade network infrastructure devices such as firewalls, routers, and switches. The configuration scenarios included industry standard technologies like VPNs, VLANs, NAT, and PAT
- Implemented a custom communications protocol in Ruby as a plugin for an end-to-end communications framework built on Ruby-on-Rails
- Participated in the full lifecycle development of these tools from requirement analysis, design, and development to testing and maintenance support
- Used an Agile Scrum approach to software development with industry standard best practices of test-driven development, unit testing, continuous integration, and revision control (subversion and git)
- Created the Online Disassembler (ODA) website and lead a team of developers to develop the Django-based backend framework of the website. Wrote the backend Python wrapper to access the native BFD shared library, which provides the disassembly engine. Integrated a prototype for dynamic runtime analysis of packed binaries into ODA by customizing and patching the open source Cuckoo virtual sandboxing platform. Assisted in website UI design and contributed new features on both the client-side and server-side code base.

INTELESYS CORPORATION

ELKRIDGE, MD Senior Firmware Engineer

JUNE 2010 – March 2015

- Conducted software security audits of electrical power critical infrastructure (SCADA) systems
- Developed proof-of-concept manipulations against these systems to demonstrate the severity of the weaknesses uncovered
- Provided guidance on security critical infrastructure systems to defend against the weaknesses identified and mitigate their risks
- Reverse engineered proprietary network protocols and developed Wireshark dissectors to parse them. Identified security weaknesses of these protocols and wrote proof-of-concept manipulations against them.
- Performed software security audits of networking devices to identify weaknesses in their booting mechanisms and runtime operations
- Designed and developed MIPS and x86 firmware for custom, embedded devices.
- Developed a Linux-based IP tunneling solution using the TUN/TAP virtual network kernel devices.
- Developed distributed processing frameworks based on RPC technologies.
- Wrote a Python utility to extract over 300 features from Windows PE executables, which were then processed by an automated malware analysis platform

EVI TECHNOLOGY / EDO CORP. / ITT CORP.

COLUMBIA, MD Senior Firmware Engineer

APRIL 2005 – JUNE 2010

- Analyzed software on Windows and a variety of embedded platforms with IDA Pro
- Designed and developed a portable architecture for an embedded, non-multitasking kernel comprising a device driver framework, network stack, and filesystem ported to a variety of embedded networking platforms.
- Developed device drivers for UARTs, compact flash devices, and network interface controllers (both wired and wireless).
- Developed system-level code for exception handling and virtual memory management.
- Developed the VHDL firmware for a CPLD used to bootstrap a Xilinx FPGA. Developed bootloaders for a variety of platforms.
- Developed Wireshark/Ethereal dissectors for custom network protocols.
- Developed and maintained the SCons (Python-based) build system used for building our cross-platform code.
- Served as team technical lead and supervisor of a small team of engineers.

U.S. DEPARTMENT OF NAVY (NAWCAD/NAVAIR)
PATUXENT RIVER, MD Computer Engineer
JUNE 2003 – APRIL 2005

- Developed a variety of Windows and VMS-based applications in various programming languages (C, C++, C#, Visual Basic) to support the infrastructure of a flight simulation laboratory.
- Wrote a 1553 bus controller in C to simulate an aircraft mission computer.
- Integrated RADAR and FLIR image generators with aircraft avionics.
- Implemented a C#- based multi-platform simulation executive.
- Designed and developed a Windows DLL to transmit real-time simulation data as a PCM data stream over fiber to a telemetry processing facility.