Curveball: Recent Windows Vulnerability

Prithika Vijayakumar
prithika@Knights.ucf.edu

Priya Sudharsanan
priya.sudharsanan@knights.ucf.edu

David Wecke
david.wecke@knights.ucf.edu

1. MOTIVATION & PROBLEM STATEMENT

The problem that we addressed is a recently patched Windows 10 vulnerability that allowed for the signing of fake signatures to be passed off as legitimate with no way to tell. The motivation behind choosing this project is that we wanted to discover what threats this vulnerability exposed because Windows is still by far the most commonly used OS and threatens the greatest amount of users that use PCs. This vulnerability is specifically interesting because the NSA issued a statement to citizens to patch this immediately and potentially exposed attackers to this opening. There might still be a large amount of systems that remain unpatched and exposing the ins and outs of this vulnerability could potentially help users that ended up being attacked through it.

The problem lies with the way that Windows 10 verifies Elliptical Curve Cryptography certificates, where it is possible to trick Windows into thinking that a personally generated certificate is actually a valid Certificate Authority(CA). This allows an attacker to sign their own certificates as a fake CA, and pass of their programs or protocols as legitimate. We are also interested in learning about Elliptical Curve Cryptography, which we are not intimately familiar with.

2. RELATED WORK

The NSA released their statement on their website regarding a patch on the day of the patch release. This included a document with the results of the investigation with some potential threats that are open because of this. The statement did not go unnoticed, and it was covered extensively by the news. This gives us many articles from people in the industry of computer security and how they viewed both the vulnerability and the NSAs involvement in a private companies software.

Related to understanding more about the technical aspects of the problem are several proof of concepts provided by users of security forums, and some more in depth discussion about the mathematical aspect of the vulnerability. We used all of the proof of concepts as blueprints to test the vulnerability ourselves. The difference between these tests and our work is that we will go a step further to analyze the dangers that this vulnerability still poses. The discussions are largely focused on what the vulnerability looks like, how to reproduce it, and what the solution to this is. We are looking at the impacts that this vulnerability could still have on older and unpatched systems running Windows 10.

We also looked at similar exploits that have happened in the past and analyzing their impact, reach, and uses. Many famous attacks have happened and been documented so we can use that for analysis in case there is not much information on our exploit.

3. WINDOWS VULNERABILITY

We conducted research in four main points of the vulnerability. What is the vulnerability? Here we research the technical definition of the vulnerability, and how it was created. This also includes, attempting to recreate the vulnerability in a secluded and safe environment to understand how it works. Why did the vulnerability arise? After conducting research on how the vulnerability works, we identified why this existed in the first place. What risks does the vulnerability pose? Through identifying how and why the vulnerability came to be, we were able to identify specific threats that are open because of it.We also looked into the details of fixed patch. After looking at what the details of the attack are and doing research on fixes already implemented, we can now tell, what the patch does in order fix the problem.

4. METHODOLOGY

The way we have carried out this project involves replication of the attack and a lot of research about this vulnerability,why it was exposed by NSA,cause of vulnerability,possible exploits with this vulnerability,different vulnerabilities of windows,identification of malicious certificates.

4.1 Technical Details of the attack and replication

CVE-2020-0601 or more commonly referred to as "CurveBall" is a vulnerability in the way Microsoft Windows CryptoAPI ensures legitimacy of certificates signed using Elliptical Curve Cryptography. This vulnerability could be exploited to allow an attacker to sign a malicious program and pass it off as a legitimate source. A rather comical proof of concept of this exploit was shown on twitter by user @saleemrash1d, where they showed a YouTube video being

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

passed off as the website NSA.gov with a valid certificate from PayPal.

The reason that this exploit is possible is because the CryptoAPI does not check all of the parameters when verifying a certificate. The problem is that it does not check for the generator parameter G, which is responsible for generating the public key of a certificate authority when verifying a certificate.

A user is able to provide their own generator for this purpose. The mathematics behind this are actually very straightforward. We get the public key of a trusted CA, and set it equal to say Q. In elliptical curve cryptography, the public key is equal to the generator point, G, times a secret key, k. Now, the main problem is that Windows will only check the public key of a certificate to ensure its legitimacy. We generate our own private key, lets call it k'. Then we create our own generator point, G'. Since we can provide our own generator and our own key k', all we need to do is set G' to be G'*k' = Q, so Q' = Q/k'.

We never need to find out the private key of a CA, and can instead created a spoofed version of it using our own generator. This exploit relies on this generator G', and therefore in order to work correctly, the spoofed CA certificate must always be provided with the certificate which has falsely been signed by it. Windows will otherwise not have the correct basis and will not know that this certificate is valid, or rather think that it is valid.

For the replication of the exploit we attempted to use multiple scripts before we had understood properly what the exploit was about, and ran into a few issues. The result of Saleem Rashid's concept was a failure because the version of OpenSSL we were using at the time for Windows was not up to date and was lacking some functionality. Additionally, certain libraries that he used in Python were not available easily for Windows. In the end ,the attack he created would need to be replicated on a Mac machine and was too impractical for use.

Using a concept provided by user ollypwn on GitHub, which used Ruby instead of Python, the proof succeeded and we was able to replicate the attack. This attack in particular used the MicrosoftECCProductRootCertificateAuthority.cer certificate authority, which is automatically trusted on Windows 10 machines. Using this the result was still negative because the exploit has since been patched but the proof stands. We were also able to generate a spoofed CA using a different certificate authority. The only requirement was that it used ECC, since of course this was the nature of the exploit.

Looking to the future is to download and run a virtual machine with an older version of Windows 10 to verify the exploit ourselves and generating a good example case of what could be done with the exploit, as currently the faked certificates are detected as such and the impact is less feasible with this result. It is also still possible to now be able to tell which avenues of attack are open to this vulnerability and can be documented and researched from this point on.

4.2 Research on CVE-2020-0601:

- **What caused this vulnerability:** The root cause of this vulnerability is a flawed implementation of the Elliptic Curve Cryptography (ECC) within Microsoft's code.

  The problem with Microsoft's vulnerable code is that it verifies certificates even if the attacker's specified their owngenerator G'and not just standard curves.

  The attacker has a new private key d' that corresponds to the original public key $Q = d * G$, which is now also equal to $d' * G'$

  This may enable attackers to specify such parameters in their certificate and sign on behalf of others to masquerade as different sites over the web.

- **Why this was exposed by NSA?** One of the main reasons for the NSA to share the vulnerability is because of EternalBlue, the leaked NSA Spy tool, which exploited a Windows bug three years ago. That flaw was present in all versions of Windows. EternalBlue was leaked to the public. Hackers made use of this to exploit most of the Windows machines as the Windows machines around the world slowly got around to patching.

  Maybe this revelation of Windows 10 vulnerability is an attempt by NSA to avoid such fiasco.

- **History of EternalBlue:**

  EternalBlue is the name of both a tool that the NSA developed to exploit the vulnerability as well as it is also the name of software vulnerability in Windows operating system. In April 2017, the exploit leaked to the public,by the mysterious group known as the Shadow Brokers.

  The tool exploits a vulnerability in the Windows Server Message Block, a transport protocol that allows Windows machines to communicate with each other and other devices for remote services and file and printer sharing. Hackers make changes to the flaws in how Server Message Block handles certain packets to remotely execute any code they want. Once they have gained access into target device, they can then fan out across a network.

  EternalBlue was the centerpiece of the worldwide WannaCry ransomware attacks that were traced to North Korean government hackers. As WannaCry hit, Microsoft took the "highly unusual step" of issuing patches for the still popular, but long-unsupported Windows XP and Windows Server 2003 operating systems.

  After this WannaCry, Microsoft and others criticized the NSA for keeping the EternalBlue vulnerability a secret for years instead of disclosing it for patching.

  EternalBlue still remains a go to tool for the hackers.Recently attackers used EternalBlue to install cryptocurrency-mining software on victim computers and servers. EternalBlue is still in demand in the world of hackers because it leaves a very limited digital trace.

- **Malicious properties in a certificate:**

  The malicious properties in a certificate can be found using software utilities like OpenSSL and Windows Certutil.

  - Certutil can be used to examine an X509 certificate by running the following command: certutil –asn<certificate-filename>
  - OpenSSL can be used to examine an X509 certificate by running the following command:

    - openssl asn1parse –inform DER –in<certificate-filename>–i –dump
    - openssl x509 –inform DER –in<certificate-filename> –text

  asn1parse is a diagnostic utility that can parse ASN.1 structure, which includes to be signed certificate, type of signature algorithm used and the bits corresponding to the signature.

- Certutil can be used to list registered elliptic curves and view their parameters by running the following commands:
    - certutil –displayEccCurve
    - certutil –displayEccCurve <curve name>
- OpenSSL can be used to view standard curves enabled/compiled into OpenSSL by running the following commands:
    - openssl ecparam –list curves
    - openssl ecparam –name<curve name> –param enc explicit –text

  The above commands can be used to view standard elliptic curves and their original parameters like generator G, private key d and the public key Q. This helps to identify malicious certificates as the attacker would have used a different generator value G' and would have created his own private key d' such that it corresponds to the original public key Q.

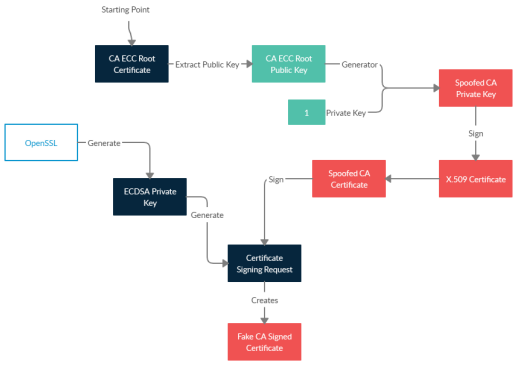
- **Consequences predicted by various parties** Different people had different views on the possible threats and effects the vulnerability poses.Few of them are listed below
    - The CERT Coordination Center, the vulnerability disclosure center at Carnegie Mellon University, stated that the vulnerability could be used to intercept and modify HTTPS and TLS communications.
    - Jake Williams, a former NSA hacker, explained to TechCrunch that the fact that the vulnerability had been reported to Windows rather than weaponized was "encouraging". Williams says that, "This one is a bug that would likely be easier for governments to use than the common hacker. This would have been an ideal exploit to couple with man in the middle network access."
    - Williams opinion about this flaw was that it could work as a type of "Skeleton-key" that would allow to gain access and control of all kids of security, facilitating the execution of malware without it being detected.
- **Possible exploits and their effects** The vulnerability or flaw in Microsoft's CryptoAPI service, allows an attacker to potentially exploit the bug by undermining how Windows verifies cryptographic trust and can enable remote code execution by taking control over the user's device. The CryptoAPI enable developers to secure Windows-based applications using cryptography and is used by developers to allow them to sign software and files or helps them to generate digital signature and certificates that are used for authentication. If this digital signatures were mishandled by an unauthorised person, then the users would have no idea on whether a file that was received was malicious as it would pose to be from a trusted provider. Apart from faking digital signatures, the attackers will be able to perform several other attacks. The vulnerability when exploited would allow attackers to defeat trusted network connections and disguise as legitimate trusted entities. Validation of trust impacts several crucial sections that are important to provide a secure system. They possible impact would be on
    - HTTPS connections
    - Signed executable code launched as user-mode processes
    - Signed files and emails

  The vulnerability exists in the way Windows CryptoAPI (Crypt32.dll) validates Elliptic Curve Cryptography (ECC) certificates. A successful exploit could also allow the attacker to conduct man-in-the-middle attacks and decrypt confidential information on user connections to the affected software.

  This vulnerability especially allows for the spoofing of Microsoft certificates that are used to establish authenticity of files and network communications. Any software that depends on the CertGetCertificateChain() function of the cryptographic library (that is responsible for ensuring that the certificate is of a recognised authority) will be unable to ensure this functionality (legitimacy of the certificate) and be misled in determining the authenticity of this certificate chain.

  In this Vulnerability, the Windows CryptoAPI $Crypt32.dll$(a windows module that handles certificate and cryptographic messaging functions in the CryptoAPI) is unsuccessful in matching the elliptic curve parameters of the given root certificate to those that Microsoft knows. Mathematical properties that are unique to elliptic curve are not exploited by this vulnerability. A similar bug can be exhibited in a Digital signature library and it is possible to show how it would behave when a normal DSA is used by Crypto32.dll.

4.3 Pipeline



Our final implementation of the exploit was done using a patched version on Windows 10, OpenSSL version 1.1.1d

and ruby version 2.6.5p114. We ended up not using a Virtual Machine to run the exploit because we did not want to exploit Windows and it was not necessary. All we wanted is to show that the exploit exists/existed at a time, and a proof of concept to understand why it did, how it worked, and what dangers it entailed. To start our exploit setup, we first require a Certificate Authority(CA) that is trusted by default in Windows 10 which uses ECC. In our version we used the Microsoft ECC Product Root Certificate Authority certificate. Next we use ruby in combination with openssl to extract the CA's public key.

$$ca = OpenSSL :: X509 :: Certificate.new(CA.crt)$$
$$ca_key = ca.publickey$$

The next part is where the exploit is used. Windows does not check the generator function properly and allows one to set their own generator function. The goal for an attacker would be to create a generator function such that their curve and private key set generate the same public key as that of a trusted CA. The exploit allows us to do this with a rather simple solution. Recall that in ECC, the private key is the number of times the dot operation is performed on G. Let us say Q is the CA's public key, k is their private key, and G is the normal generator function. Since both Q and G are points on the curve, we are able to set our G' to be Q. Finally, we simply set our private key to be one. That way no dot operation is ever performed and our public key matches that of the CA. Again, we do this using openssl and ruby.

$$Q' = k'dotG' = 1dotQ = Q$$
$$ca - key.private - key = 1$$
$$group = cakey.group$$
$$group.setgenerator(ca - key.public - key, group.order, group.cofactor)$$

We have our fake private key with a bad generator function now, now we sign a new certificate with it. This will be our fake CA, we are going to be impersonating the CA here. This is done using OpenSSL through command prompt.

$$opensslreq - new - x509 - keyfake - ca.key - outfake - ca.crt$$

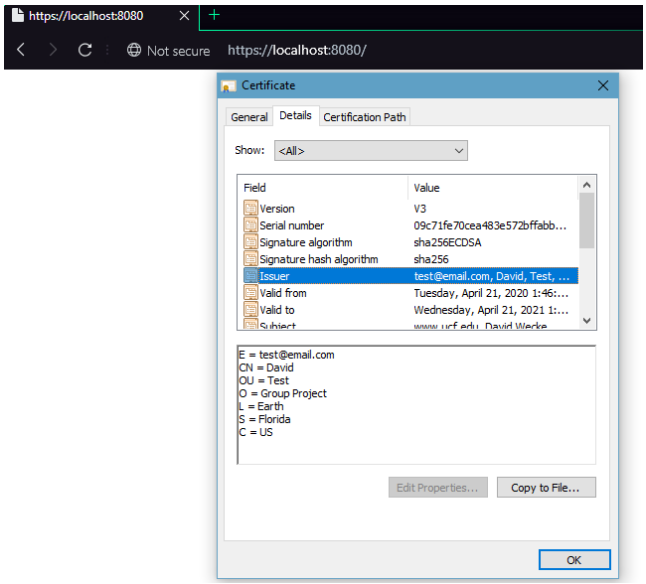
Next we generate a key for use with whatever we are trying to sign, and then create a certificate signing request(CSR). The openssl.conf file contains data for the questions that would ordinarily be asked when creating a CSR such as the common name and certificate usage options. We will be using this certificate for server authentication.

$$opensslecparam - namesecp384r1 - genkey - noout - outserver.key$$
$$opensslreq - new - keyserver.key - out server.csr - configopenssl.conf - reqextsv3_tls$$

For the last step of the exploit we now sign our own CSR, pretending to be the CA we initially spoofed.

$$opensslx509 - req - inserver.csr - CAfake - ca.crt - CAkeyfake - ca.key - CAcreateserial - outserver.crt - days365 - extfileopenssl.conf - extensionsv3_tls$$

That was it, we now have our private key, associated signed certificate, and a fake CA certificate. In our implementation we used Node.js to serve up a local server with our three files specified. This does not show up as a secure server anymore, but we were out to find out only how this attack would be carried out. Instead of specifying that we wanted to use our certificate for sever and client authentication, we could have also chosen to use it for code signing, which would allow an attacker to sign malicious software and Windows not being able to tell since the certificate it is signed with matches the public key of the certificate found in the cert store.



The attack was scarily easy to replicate with two very scary possibilities: server authentication and code signing. These could be used in conjunction with each other to serve up a fake trusted server which hosts a malicious file sign by the same fake certificate. In the end we were unable to find any accounts of this attack being used, but we have no doubt with an attack this simple to carry out, someone will have been attacked by it. The attack compromises integrity by faking the CA that we rely on to prove our identity to others.

5. MANTIFESTION IN A DSA ALGORITHM

The vulnerability exists only in the elliptic curve cryptographic functionality and interestingly it does not exploit any mathematical properties unique to the elliptic curve. Hence it is possible that the exact same bug can manifest itself in a normal DSA signature verification library.Below we show how this vulnerability could have affected a normal DSA if the Crypto32.dll was used.
To set up DSA, a user will need a prime p and a generator g. These two parameters are used by the users to generate a private key x and a public key

$$p_k = g^x mod\ p$$

.Only the private key can used to create the signature while the public key is used to verify it.This way of signing makes the signature forgery process very hard.
But these Digital signature algorithms have few disadvantages.They do not have any mechanism by which a user

can trust a given public key associated with a specific entity.Hence the X.509 certificates were used.The X.509 cert is a file that specifies that a particular public key belong to a particular person signed by another person.Therefore every person signing a document will have a X.509 certificate.These certificates could be chained together,beginning at the "root" certificate that attests the identity of a root certificate authority.This root certificate authority signs the certificate under him or the intermediate certificates attesting the identity of the intermediate certificate authority, these intermediate certificate authority signs other intermediate certificate authority attesting their identity and this process goes on until the leaf certificate at end is reached.
Each of this X.509 certificate contains information that includes the signature algorithm and the corresponding parameters used.A typical Microsoft's certificate would contain the following(the below is a heavily simplified structure):

- Certificate Authority:Microsoft
- Name :name of owner
- Public key Information
  - Algorithm:DSA
  - Prime: p
  - Generator:g
  - Public key:P_k

Now whenever a Windows user receives a X.509 certificate chain, they check to verify that the certificate authority at the root of this chain is someone that Microsoft trusts. But due to this vulnerability ,the Windows only checks to verify if the public key of the certificate that was received matches to a trusted entity and does not verify the security parameters.In such case the attacker would be able to modify the value of p or g which are associated with the public key P_k without the Windows knowing it.This way an attacker would be able to gain access to the users system thereby breaking the security of DSA.
To carry out such an attack,the attacker could simply set g=p and make private key x=1.This way the attacker would be able to sign any message as if he/she was the legitimate owner of the public key P_k .This is because the attacker knows the private key $x = 1$. Another simple exploit of this vulnerability would be to select a private key x' and set the malicious generator $g' = x'^{-1} * P_k$.Here it can be seen that the certificate still has a secret key that is not known to the original owner and is known only by the attacker.
Essentially, if the authenticity of parameters associated with a given public key is not established, the attacker can choose any private key they want.

6. RESEARCH ON OTHER WINDOWS VULNERABILITIES

During the course of this project, we learnt a lot about different vulnerabilities in Windows OS like the BlueKeep,Nimda, Heartbleed,Shellshock and so on. One of these vulnerabilities is discussed in detail below:
BlueKeep (CVE-2019-0708): It is a vulnerability present in Windows in its Remote Desktop protocol (RDP) implementation that allows remote code execution.This vulnerability is present in Windows 7, Windows XP, Server 2003 and 2008. Windows 10 is safe from this vulnerability.Microsoft has issued a patch,but still large number of machines are still vulnerable. The possible exploits that can take place due to this vulnerability is denial of service and using this vulnerability to execute ransomware attack.
Measures taken to prevent this:

- Block TCP Port 3389 as it is used in RDP protocol and will block attempts to establish a connection.
- Network level authentication is enabled
- Remote Desktop services can be disable if not used.

7. EVALUATION

We focused our attention on this project into three main points: research, replication, and analysis. Due to the NSA announcement there was a large amount of media attention and many articles were written. We read many of these looking for details about the attack and the impact it had. We also searched around forums and blogs focused around hacking that covered the attack, with possible examples. A lot of information was available on sites that specifically cover vulnerabilities when they are revealed. For replication we were able to find several proof of concept attacks, but not all of them were functional on Windows, which is what our systems are. For the proof of concepts, we required the installation of certain security tools such as OpenSSL and DigiCert's certificate signing request generator. Finally, we analyzed the information that gathered from research and the replication, looking for what damages were done and what methods of attach the exploit allows for. This led us to the following evaluation metrics that we wanted to answer for our project:

- How many attacks using curveball did we identify, and how dangerous are they?
- Have these attacks been used in the wild?
- Can we find any mitigation methods for these?
- Is the attack reproducible, and if so, how difficult is it?

7.1 Evaluation Results

We were able to identify four different types of attacks using the vulnerability. An attacker could compromise an HTTPS connection using a fake certificate, like in the attack outlined previously in this paper. Using this same method an attacker could also fake code signing, email signing, and signing files. The exploit could also be used in a man in the middle attack. We were not able to find any evidence of any of these attacks being used in the wild, however we believe the possibility is high that someone has been attacked but it was never publicly documented. There might also still be some machines that are at risk, such as companies that were unable to upgrade because they have many machines. The mitigation for these machines and anyone in general is patching using the Windows updater, using proxies for data transfer that have additional TLS inspection, analyzing packet traffic through something like Wireshark, and analyzing the certificates attached to anything that requires it using OpenSSL. Firefox for example was not vulnerable to the attack because they do not use Window's CA store

to check TLS certificates, and the curveball signed signatures were found to be malicious. It comes down to finding out either whether the certificates are fake either manually through the tools, or by having someone not compromised check certificates. We were also able to replicate the attack and it was very easy to do. In conclusion, we found that most machines are safe and the impact was very low, but the severity or potential for attack was very high.

8. MITIGATION

On January 14th 2020,Microsoft published a security advisory and the patch for the Vulnerability CVE 2020-0601.NSA recommends installing this patch in all windows 10 and Windows server 2016/2019 systems and this is the best solution to mitigate this vulnerability Systems that are high risk of exploitation includes

- Windows based web application, web server and proxies that perform any kind of TLS validation.
- Systems that host critical infrastructure such as DNS server, VPN server, update servers etc
- System used by privileged users.
- Essentially all systems that are connected to the internet

It is highly important for these system owners to prioritize patching the endpoints.
The patch includes a couple of checks during the signature verification to make sure that the elliptic curve digital signature algorithm parameters are authenticated.There are network devices and endpoint system logging features that could detect any potential vulnerabilities like these.One such was the Fortinet firewall that was able to detect the vulnerability(exploits) even before the patch was released. Hence installing and constantly updating such firewalls and Antivirus defenders are proven to mitigate any exploits that might have been caused by the vulnerability.

Patch details
The patch released on January 14th by Microsoft, includes patches to 17 other new vulnerabilities in addition to CVE-2020-0601 (Curveball).Microsoft added a logging mechanism, which tracks attempts to exploit this vulnerability.The patch included the typical release of operating system patches with the addition of a new Application Programming Interface function. The new CveEventWrite function will be able to publish events when there occurs an attempt to exploit security vulnerabilities in user -mode applications. The analysts will gather the alerts on the application message "CVE-2020-0601" as a means to capture any attempted exploitation of this vulnerability on the patched systems.In addition to CVE-2020-0601 the patch also fixes two other major security related vulnerability: CVE-2020-0609 and CVE-2020-0610.Hence it is very important to install this patch.

9. EXPECTED OUTCOMES AND RISK MANAGEMENT

Replicating the attack was difficult, but manageable. The certificates should not come up as legitimate any more, so the replication should fail. We learnt about how the attack is created, which gave us some insight into what avenues of attack are possible and where the attack could still potentially harm users. This attempt to replicate the attack helped us understand more about ECC.
Researching about this vulnerability gave us an insight of how usually a vulnerability occurs, what steps will be taken if such vulnerability arises,severity of the vulnerability and how this is fixed.

The risk is that there is not much information on our current exploit because it has already been patched and much of the hype surrounding it is due to the NSA announcement. Here we already have the announcement itself and some technical details about the attacks so it should be possible to estimate what the impact could have been.

10. PLAN AND ROLES OF COLLABORATORS

David worked on replicating the attack and learning about the technical aspects involved with this exploit. He focused in on how ECC works, and why the exploit was possible in terms of the technical aspect. He analyzed the results of the attack on a patched machine, as well as attempted to use an older version of Windows 10 in a virtual machine to successfully execute the attack. He created a set of code that is able to replicate the attack.
Prithika explored on the possible threats and exploits this vulnerability poses and has also provided details on the effects and mitigation for the vulnerability.She has also provided technical details and working on how a similar bug could have been manifested itself in a normal DSA signature verification library.The patch details that fixes this vulnerability is also provided by her. Furthermore a small report on different views of the vulnerability by different people has also been provided by her.
Priya dealt with the research about the vulnerability that includes, what was the root cause of this vulnerability,what was the reason behind NSA's revealing of this vulnerability and how the vulnerability works.She also analysed different ways to find the malicious property in the certificate.Her other works include the comparisons of different vulnerabilities of Windows.

11. REFERENCES

- https://github.com/saleemrashid/badecparams
- https://packetstormsecurity.com/files/155960/CurveBall-Microsoft-Windows-CryptoAPI-Spoofing-Proof-Of-Concept.html
- https://packetstormsecurity.com/files/155961/CurveBall-Microsoft-Windows-CryptoAPI-Spoofing-Proof-Of-Concept.html
- https://arstechnica.com/information-technology/2020/01/researcher-develops-working-exploit-for-critical-windows-10-vulnerabilily/
- https://github.com/ollypwn/CurveBall
- https://media.defense.gov/2020/Jan/14/2002234275/-1/-1/0/CSA-WINDOWS-10-CRYPT-LIB-20190114.PDF
- https://www.wired.com/story/nsa-windows-10-vulnerability-disclosure/

- https://blog.trailofbits.com/2020/01/16/exploiting-the-windows-cryptoapi-vulnerability/
- https://www.pandasecurity.com/mediacenter/news/critical-windows-10-vulnerability/